Sheaf Theoretic Models for Routing in Delay Tolerant Networks

Robert Short and Alan Hylton and Jacob Cleveland
NASA Glenn Research Center

Michael Moy
Colorado State University

Robert Cardona and Robert Green and Justin Curry
University at Albany - State University of New York

Brendan Mallery
Tufts University

Gabriel Bainbridge
The Ohio State University

Zander Memon
American University

***Abstract*—One key to communications scalability is routing; as such the goal of this paper is to build upon successful efforts towards general routing for space-based networks. With the ever-increasing accessibility of space, the number of assets is increasing, which becomes a critical communications burden in terms of scheduling, spectrum allocation, and resource allocation. In order to mitigate these concerns, a true networking approach is necessary; a standard approach for space systems is Delay Tolerant Networking (DTN). For DTN to be a meaningful answer to the Solar System Internet (SSI) question, DTN must offer meaningful routing solutions that span the heterogeneous collection of links and nodes. This, in turn, depends on the general structure of these disconnected networks – a structure that remains largely unknown.**

In ground communications networks, routing decisions are made based on several pathfinding algorithms working in tandem. In previous work, we modeled Dijkstra's pathfinding algorithm using sheaves and provided a more general framework for determining paths using sheaves over graphs. Continuing our sheaf-theoretic approach, we introduce here an expansion of our pathfinding sheaf to handle more general information, and we expand on additional pathfinding algorithms that can be represented using sheaves. Moreover, we demonstrate means of combining multiple algorithms into a single sheaf structure so that changes of scale can be presented in the language of sheaves.

In addition, space communications networks rely upon radio transmitter antennas which can establish broadcast and multicast communications options, rather than the primarily unicast options available to wired networks. Last year, we also introduced a multicast routing sheaf for presenting broadcast, unicast, and multicast communications over a graph. Extending that work, we also introduce queuing sheaves so that we can blend these communications options together to simulate a variety of routing options across space networks. In addition, we include examples to illustrate the applicability of this abstract theory to routing in disconnected networks.

1. INTRODUCTION

In 2016, NASA took a step towards the Solar System Internet (SSI) concept by deploying Delay Tolerant Networking (DTN) on the International Space Station (ISS) [1]. By working towards an operational DTN capability, several questions regarding communication in space started to get answers. Among these are how to achieve a returns-to-scale with the amount of communicating assets, which is ever-growing [2]. The existing manual scheduling approach to communications does not scale, and indeed NASA has noted that scheduling communications across a network can take up to five days [3].

DTN begins to mitigate this as an overlay network, whose goal is to integrate otherwise disparate network segments. The inherent challenges of realizing the SSI include the diversity of these segments: some assets are mobile, the latency might be less than a second or more than an hour, and the overall network might always have multiple connected components (i.e., it may never feature end-to-end connectivity at any given time). To overcome this, several routing algorithms have been devised to act upon various DTN daemons, such as Contact Graph Routing (CGR) [4], which relies on schedules; Delay Tolerant Link State Routing (DTLSR) [5]; and Probabilistic Routing Protocol using the History of Encounters and Transitivity (PRoPHET) [6] – the latter two featuring discovery. Note that discovery may not always be an option. Interplanetary distances and the resulting propagation delays can lead to discovered devices being unavailable before communication can occur.

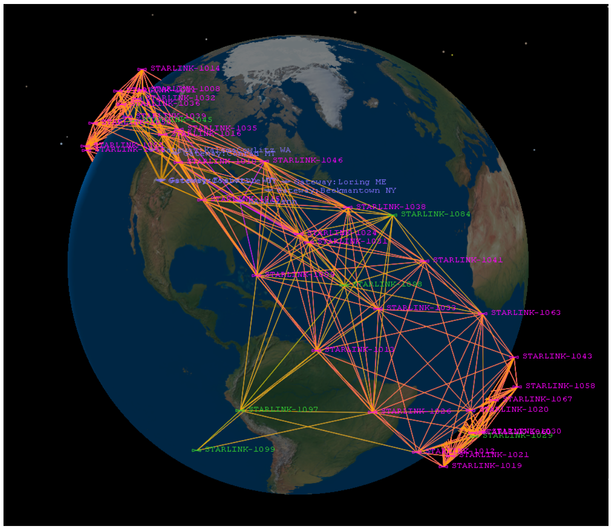


Figure 1. Sampling of the Starlink network

Sheaves formalize how one can glue local observations together into global data – consistently and uniquely – or why this would fail. In a network, connecting a laptop to Wi-Fi is a local phenomenon that gives rise to paths (routes) across the Internet – a global phenomenon. In Figure 1, we consider a sampling of the Starlink network[1], and realize a similar action must take place. That is, as the space network changes, so do the possible routes, and from the perspectives

[1] See https://www.starlink.com/.

of the individual actors there are likely differing notions of the global picture. This would especially be true over interplanetary distances. In addition, as space networks grow in size – such as the mega constellations developed by Starlink – routing based on understanding the full scope of the network may become infeasible or unwieldy from a computational standpoint, a distribution standpoint, or a memory standpoint. For this reason, a structure prescribing local phenomenon that can be verified to be consistent globally is needed, and sheaves provide such a structure.

It has been noted that the approaches of CGR, DTLSR, and PRoPHET, while very useful in their own rights, do not rest upon strong mathematical foundations [7], [8]. The authors have discovered that the mathematical theory of sheaves – explained in Section 2 – provides much of the needed machinery to not only describe these routing algorithms, but also to describe how they work together: this is analogous to how Dijkstra and Bellman-Ford work together (so to speak) in the Internet. In fact, by generalizing these familiar algorithms and constructs to the more general language of sheaves, we can begin applying them to more general structures, such as temporal graphs.

The idea of building sheaf-theoretic models for DTN applications has been explored within the High-rate Delay Tolerant Networking (HDTN) project at the NASA Glenn Research Center (GRC) for the past several years. Our first venture into this concept came in a paper to this conference [7] where we explored some of the temporal aspects of DTN connectivity and discussed the what and why of sheaves. Last year, we provided some initial development of sheaf-theoretic models for routing in DTN through [9], and we continue that work in this paper. We hasten to add that while an appeal to sheaves represents a large step towards the abstract, the fruits of the labors remain readily implementable. Indeed, sheaves can be realized in software – e.g. through the Python library PySheaf [10][2] – and they have also been leveraged to develop novel star tracking algorithms [11].

This paper builds off of previous successes, such as the modeling of Dijkstra's algorithm in the language of sheaves, to include other "sheafy" approaches towards building a better DTN theory. After we recall the intuition and the definition of a sheaf in Section 2 and provide a brief reminder of previously defined networking sheaves in Section 3, we move on to novel material. The most familiar starting point is that of the *tree sheaf* in Section 4, which encodes trees in the language of sheaves. As we can think of trees as a gluing together of paths, we examine the same gluing action applied to sheaves in Section 5. Following the notion of a limit to products in Section 6, we explore how various routing algorithms can operate on the same network when represented as sheaves. In Section 7, we consider applications of *zigzag persistence*, a tool from computational topology that is well-suited for temporal systems. In Section 8, we turn our attention to sheaves as an organizational framework to enable the usage of machine learning to make routing decisions. We end with final remarks and ideas for future work.

2. Sheaves on Posets

Sheaves are considered by some authors to be the canonical data structure for data integration [12]. If one has observations over two different portions of a system, one would like to glue these observations together consistently, and sheaves provide a way of doing this. Although in most math textbooks, e.g. [13], [14], [15], sheaves are defined as an attachment of a piece of data to an arbitrary (open) subset of a system and consistency is imposed as a requirement, recent applications of sheaf theory outside of mathematics [16] have advanced the perspective that one should define a sheaf in terms of the minimal data required to generate consistent observations. This is done by describing a sheaf in terms of transformations of data across pairwise relations in a poset [17], which we use to model local interactions of a system, as in a directed graph. Assignment of data to larger portions of the system can be accomplished via an auxiliary construction, known as a Kan extension [18]. With this in mind, we provide a brief introduction to sheaves, giving a description that fits the examples in this paper. A more in-depth explanation of sheaves their applications can be found in [19]. General background information on category theory can be found in [20], [21].

Definition 2.1. A **sheaf on a poset** $(P, \leq)$ is a covariant functor $\mathcal{F} : P \to \mathbf{Set}$. In more detail, this entails

- To each $x \in P$ a set $\mathcal{F}(x)$, called the **stalk** at x is assigned,
- To each pair $x \leq y \in P$ a map of sets, called the **restriction along** $\boldsymbol{x \leq y}$, is specified: $\mathcal{F}(x \leq y) : \mathcal{F}(x) \to \mathcal{F}(y)$.
- Finally, functoriality requires appropriate composition: If $x \leq y \leq z$ is a comparable trio, then $\mathcal{F}(y \leq z) \circ \mathcal{F}(x \leq y) = \mathcal{F}(x \leq z)$.

Dual to this notion is a **cosheaf on a poset** P. This is a functor $\mathcal{G} : P^{\mathrm{op}} \to \mathbf{Set}$. This means

- To each $x \in P$, a **costalk** $\mathcal{G}(x)$ is assigned, and
- To each $x \leq y \in P$, a map, called the **extension along** $\boldsymbol{x \leq y}$, is specified $\mathcal{G}(x \leq y) : \mathcal{G}(y) \to \mathcal{G}(x)$, subject to
- If $x \leq y \leq z \in P$, then $\mathcal{G}(x \leq y) \circ \mathcal{G}(y \leq z) = \mathcal{G}(x \leq z)$.

Remark 2.2. What we have defined above are properly called **(co)sheaves of sets**. Sets are a relatively unstructured data type. If one wanted to assume a vector space structure on our data, one would replace $\mathbf{Set}$ with $\mathbf{Vect}$ in the above definition and add the appropriate adjectives where necessary, i.e. replace "maps" with "linear maps" above. A similar remark holds for groups or other categories of data types.

Typically, sheaves and cosheaves are defined on open sets of a topological space. Sheaves are contravariant with respect to inclusion of open sets and cosheaves are covariant with respect to inclusion. This explains why the maps above are called restriction and extension maps, respectively: in a sheaf, one restricts from a larger open set to a smaller open set, whereas cosheaves have the opposite behavior. We now describe what these open sets look like for a poset P.

Definition 2.3. Let $(P, \leq)$ be a poset. An **up set** is a subset $U \subseteq P$ such that if $x \in U$ and $x \leq y$, then $y \in U$. A **principal up set** is an up set of the form $U_x := \{y \mid x \leq y\}$. Up sets are also called **Alexandrov opens** because they define a topology. Principal up sets define a basis for the Alexandrov topology.

Nearly all of the posets considered in this paper come from graphs, whereas the vertices and edges are regarded as points

[2] See https://github.com/kb1dds/pysheaf.

in a poset. Graphs are instances of cell complexes, so the language of cellular sheaves is often used; these were first introduced by Shepard [22].

Definition 2.4. Let G be a (multi)graph, i.e. several edges can connect the same pair of vertices, but an edge connecting a vertex to itself is not allowed. Let $P_G = V \cup E$ be the union of the vertices and edges, which are both examples of **cells**. P_G carries the structure of a poset by declaring $v \leq e$ if the vertex v is incident to the edge e.

1. A **cellular sheaf on G** is a functor $\mathcal{F} : P_G \to \mathbf{Set}$.

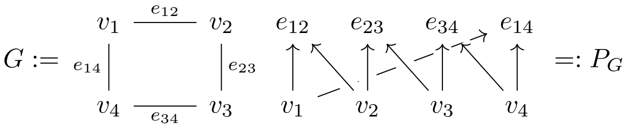
2. A **cellular cosheaf on G** is a functor $\mathcal{G} : P_G^{\mathrm{op}} \to \mathbf{Set}$.

We now explain why this definition makes sense when using the traditional open set definition of a sheaf. Associated to each vertex v in a graph G is its **open star**, which is the same thing as the principal up set at v, i.e.

$$U_v := \mathrm{star}(v) = \{v\} \cup \{e : e \text{ is incident to } v\}.$$

The principal up set at an edge e is just the edge itself, i.e. $U_e = \{e\}$. The restriction map $\mathcal{F}(v \leq e) : \mathcal{F}(v) \to \mathcal{F}(e)$ can then be viewed as a map that restricts data from U_v down to the subset U_e by defining $\mathcal{F}(U_v) := \mathcal{F}(v)$ and $\mathcal{F}(U_e) = \mathcal{F}(e)$.

Example 2.5. The cycle graph $G := C_4$ has the following poset structure P_G,



A cellular sheaf $\mathcal{F}$ on this graph would then assign a set to each vertex and to each edge as well as maps from each vertex to all incident edges. $\mathcal{F}$ could assign to every cell the one-point set $\{*\}$, which would be an example of a **constant sheaf**, but the power of sheaf theory is in allowing the set to vary as we move across the graph.

When graphs are used to model networks, there will inherently be data associated to the vertices and/or edges of the graph. A sheaf provides a model for the storage of such information when there are some requirements on how the information in adjacent cells must be related – these requirements are imposed by the restriction maps of the sheaf, and a consistent set of data across all cells is a global section.

Definition 2.6. Given a cellular sheaf $\mathcal{F} : P_G \to \mathbf{Set}$ over a graph G, a (discontinuous) **section** is any map

$$\sigma : P_G \to \coprod_{g \in P_G} \mathcal{F}(g).$$

This means that we select an element from the set $\mathcal{F}(g)$ and assign it to each cell (vertex or edge) g. Such a section is said to be **continuous** if whenever $v \leq e$ we have

$$\mathcal{F}(v \leq e)\sigma(v) = \sigma(e),$$

i.e. the choice of element $\sigma(v) \in \mathcal{F}(v)$ is consistent with the choice of element $\sigma(e) \in \mathcal{F}(e)$ once the restriction map $\mathcal{F}(v \leq e)$ is applied to it. It is convention to ignore discontinuous sections entirely, so that when we say "section," we will always mean a continuous section. We note that not every sheaf admits a section, as Figure 2 in Section 8 illustrates.

In the context of networking, when we interpret the data $\mathcal{F}(v)$ over a vertex v as the data stored by a specific asset in the network, the data on the edges can represent data shared between two adjacent vertices. The restriction maps describe how data from a vertex is shared into an incident edge. A continuous section then represents an assignment of data to the vertices that is consistent when shared along edges.

For an alternative example, a routing algorithm may associate data to the vertices and edges of a graph and manipulate or update this data over the course of the algorithm. We may model this scenario with a sheaf as well, where the restriction maps can require that the data is not arbitrary, but is related between adjacent vertices in a way that is useful to the algorithm. Examples of such behavior is given in Sections 3 and 4. For now, we formalize the notion of globally consistent data in a sheaf.

Definition 2.7. The set of **global sections** of a cellular sheaf $\mathcal{F}$, written $\mathcal{F}(G)$, is the collection of all continuous sections, i.e.

$$\mathcal{F}(G) := \left\{ \sigma : P_G \to \coprod_{g \in P_G} \mathcal{F}(g) \;\middle|\; \sigma \text{ is continuous} \right\}$$

This can be described purely in terms of choices of section values over vertices that cohere upon extension over adjacent edges. Said categorically, $\mathcal{F}(G)$ is the equalizer of the following pair of maps:

$$\begin{tikzcd} \prod_{v \in V} \mathcal{F}(v) \arrow[r, shift left, "\delta^+"] \arrow[r, shift right, "\delta^-"'] & \prod_{e \in E} \mathcal{F}(e) \end{tikzcd}$$

To define δ^+ and δ^-, we assume a choice of orientation on the edges so that each edge e can be regarded as a pair $(s(e), t(e))$ where $s(e)$ is the source of e and $t(e)$ is the target of e. The maps in the equalizer can be now defined as follows:

$$\delta^+\big((x_v)_{v \in V}\big) = \big(\mathcal{F}(v \leq e)(x_v)\big)_{e \in E, t(e) = v}$$

$$\delta^-\big((x_u)_{u \in V}\big) = \big(\mathcal{F}(u \leq e)(x_u)\big)_{e \in E, s(e) = u}$$

$\mathcal{F}(G)$ can now be expressed as the subset of the product

$$\left\{ (x_v)_{v \in V} \in \prod_{v \in V} \mathcal{F}(v) \;\middle|\; \begin{array}{l} \mathcal{F}(s(e) \leq e)(x_{s(e)}) = \\ \mathcal{F}(t(e) \leq e)(x_{t(e)}) \\ \text{for every } e \in E \end{array} \right\}.$$

It can be shown that this definition does not depend on the orientation of the edges, as this is just a convention for computing the categorical limit of the functor $\mathcal{F}$; see §2.2.1 of [19]. When this equalizer is performed over an Alexandrov open subset U of P_G, then the resulting set $\mathcal{F}(U)$ is called the set of **local sections** over U.

We will provide more efficient algorithms for computing global sections in the context of networking in Section 8.

The Weight Sheaf

As an initial example of a cellular sheaf, we construct a sheaf representing the weight information present on a weighted graph. Let $G = (V, E)$ be a graph with weight function $w : E \to \mathbb{R}$. One usually thinks about this information living over the edges of the graph G. However, in truth, the data for the weights is not inherently stored in the edges: it is more useful to think of it stored over the vertices. So a natural question might be: how do we realize weight information at each vertex?

The following sheaf will give us insight into how this information is stored locally at each vertex in the network. We will call this sheaf $\mathcal{W}$ the **weight sheaf** for G. For $\mathcal{W}$, the information over the edges is clear: each edge should correspond to a real number, so we have

$$\mathcal{W}(e) = \mathbb{R}.$$

Then, since we will map from each vertex onto an edge using a restriction map, we require real number information for each edge incident to a given vertex v. Recall that the degree $\deg(v)$ is the number of edges incident to v. We define

$$\mathcal{W}(v) = \mathbb{R}^{\deg(v)}$$

accordingly. The final information we need for our sheaf is the restriction maps. Here, the idea is to map each number over a vertex to the number associated to the edge represented by its place in the ordering of components in $\mathbb{R}^{\deg(v)}$. For this, we assume the edges incident to v are in the order $e_1, \ldots, e_{\deg(v)}$, so that we can define the restriction maps by

$$\mathcal{W}(v \leq e_i)(x_1, \ldots, x_{\deg(v)}) = x_i$$

which is simply the projection map from $\mathbb{R}^{\deg(v)}$ onto the proper coordinate for each edge.

This gives us a complete definition of the weight sheaf associated to a graph G. For this paper, we will define our sheaves more concisely in the following format:

Definition 2.8. Let $G = (V, E)$ be a graph. We define the **weight sheaf** $\mathcal{W}$ of G so that for each vertex $v \in V$,

$$\mathcal{W}(v) = \mathbb{R}^{\deg(v)},$$

and for each edge $e \in E$,

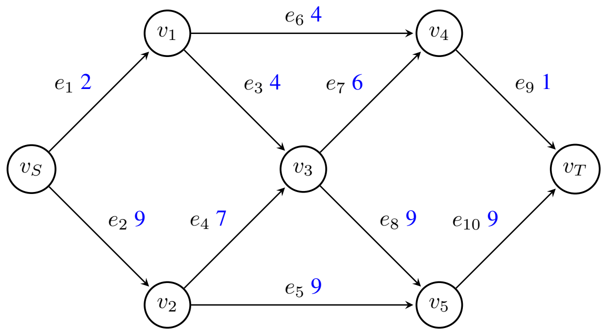
$$\mathcal{W}(e) = \mathbb{R}.$$

Finally, for each vertex v, assume the edges incident to v are ordered by $e_1, \ldots, e_{\deg(v)}$. Then the restriction maps are given by

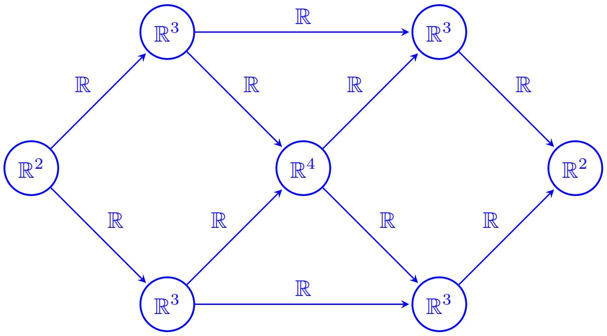
$$\mathcal{W}(v \leq e_i)(x_1, \ldots, x_{\deg(v)}) = x_i$$

as the projection map onto the corresponding coordinate.

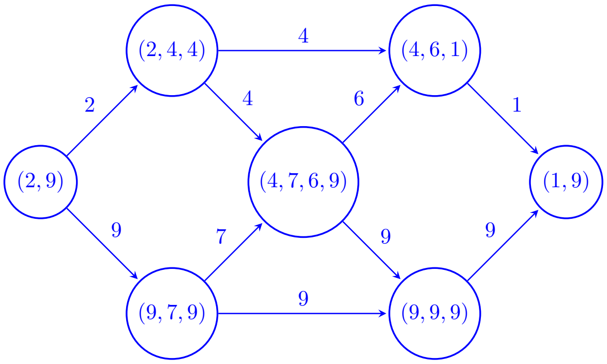
As an example, consider the graph below with weight function defined by the numbers listed next to the edge labels.



If we represent the weight sheaf of this graph visually, we have the following image.



Here we see the distinction between a sheaf and a section laid bare. This is analogous to the distinction between a data structure and an instance of data. The weight sheaf provides the structure of weight data across the network, but it does not provide the actual weight functions themselves. Instead, we can represent any weight function as an element of $\mathcal{W}(G)$, the set of sections of $\mathcal{W}$ over G. In fact, the weight function given in the definition of our graph can be represented as the following section w of our weight sheaf $\mathcal{W}$ over G.



Intuitively, we retrieve the action of the weight function $w : E \to \mathbb{R}$ from the definition of G as a section of $\mathcal{W}$, since we can see from examples that $w(e_1) = 2$ or $w(e_8) = 9$. However, the sheaf captures additional information, such as $w(v_S) = (2, 9)$, meaning we can query the vertices for the information that they have available to them and identify information relevant to local routing decisions.

3. Previous Networking Sheaves

Last year at this conference, we introduced several sheaf-theoretic structures for describing networking ideas. In this section, we present the definitions of sheaves from [9] and [23] briefly so that we can build on them later in this paper. Further details and results can be found in [9], [23], [24].

The Path Sheaf

Let $G = (V, E)$ be a finite directed graph, or digraph, with a specified source vertex v_S and target vertex v_T. We will let $\mathrm{In}(v)$ be the set of edges coming into v and $\mathrm{Out}(v)$ be the set of edges pointing away from v. The sheaf defined below has global sections corresponding to every path in G from v_S to v_T. Even though Definition 2.7 pointed out that the choice of orientation does not affect the computation of global sections, the directionality of G is programmed into the stalks of the sheaf $\mathcal{P}$.

Definition 3.1. We define the **path sheaf** $\mathcal{P}$ on G by assigning to each vertex the sets

$$\mathcal{P}(v) = \begin{cases} \mathrm{Out}(v) & \text{if } v = v_S \\ \mathrm{In}(v) & \text{if } v = v_T \\ (\mathrm{In}(v) \times \mathrm{Out}(v)) \cup \{\bot\} & \text{otherwise} \end{cases},$$

and assigning to each edge the set

$$\mathcal{P}(e) = \{\bot, \top\}.$$

If $v = v_S$ or v_T, define the restriction map

$$\mathcal{P}(v \le e)(e_i) = \begin{cases} \top & \text{if } e = e_i \\ \bot & \text{otherwise} \end{cases}.$$

If $v \neq v_S, v_T$, define the restriction map

$$\mathcal{P}(v \le e)(x) = \begin{cases} \top & \text{if } x = (e_i, e_o) \text{ and } e = e_i \text{ or } e_o \\ \bot & \text{otherwise} \end{cases}.$$

In a section, we interpret edges assigned to $\top$ as "on" and edges associated to $\bot$ as "off". Also, for each path from source to sink, there exists a global section in which exactly the edges in the path are assigned to $\top$. However, other global sections may exist: one deficiency of the path sheaf is that a section may have loops with edges assigned to $\top$ (see [23] for details). This deficiency is not present in the following modified version of the path sheaf.

The Distance Path Sheaf

In [9], [23], our goal was to understand the structure needed for Dijkstra's algorithm in terms of a sheaf. While the path sheaf $\mathcal{P}$ captures all of the paths, we wanted to incorporate weights into the sheaf structure. This resulted in the sheaf defined here.

Definition 3.2. Let $G = (V, E)$ be a digraph with prescribed source vertex v_S and target vertex v_T. In addition, suppose we have a weight function $w : E \to \mathbb{R}^+$ assigning weights to each edge. We define the **distance path sheaf** $\mathcal{DP}$ on G by assigning to each vertex the set

$$\mathcal{DP}(v) = \begin{cases} \mathrm{Out}(v) \times \{0\} & \text{if } v = v_S \\ \mathrm{In}(v) \times \mathbb{R}^+ & \text{if } v = v_T \\ (\mathrm{In}(v) \times \mathrm{Out}(v) \times \mathbb{R}^+) \cup \{\bot\} & \text{otherwise} \end{cases}$$

and assigning to each edge the set

$$\mathcal{DP}(e) = \mathbb{R}^+ \cup \{\bot\}$$

accordingly. For the source, define the restriction map

$$\mathcal{DP}(v_S \le e)(e_i, 0) = \begin{cases} w(e) & \text{if } e = e_i \\ \bot & \text{otherwise} \end{cases}.$$

But, for the target, define the restriction map

$$\mathcal{DP}(v_T \le e)(e_i, x) = \begin{cases} x & \text{if } e = e_i \\ \bot & \text{otherwise} \end{cases}.$$

For $v \neq v_S, v_T$, define the restriction maps by

$$\mathcal{DP}(v \le e)(\alpha) = \begin{cases} x & \text{if } \alpha = (e_i, e_o, x) \text{ and } e = e_i \\ x + w(e) & \text{if } \alpha = (e_i, e_o, x) \text{ and } e = e_o \\ \bot & \text{otherwise} \end{cases}.$$

This restriction map has the effect of accruing the weight of traversal across the outgoing edge. Intuitively, the only consistent sections will be choices of the weight coordinate that match the sum of the weights traversed in a path.

Multicast Sheaves

The third sheaf introduced in [9] models multicast capabilities for routing decision-making. We describe further work with this sheaf, including implementation, in Section 8, so we will recall the definition in that section.

4. The Tree Sheaf

We now provide the first novel construction of this paper: the *tree sheaf*. Like the path sheaf and distance path sheaf of the previous section, the tree sheaf allows us to reinterpret the graph theoretic concept of a tree in the language of sheaves. Specifically, sections of this sheaf will represent trees rooted at a designated source vertex. This will provide an example of how sheaves can describe data stored on the vertices and edges of a graph, as is done in routing algorithms.

As in Section 3, we assume that $G = (V, E)$ is a finite directed graph. We will make use of the notation $2^{\mathrm{Out}(v)}$, the power set of the set of outgoing edges. We assume a weight function $w \colon E \to \mathbb{R}^+$ is provided.

Definition 4.1. Let $G = (V, E)$ be a finite directed graph with distinguished source vertex v_S and accompanying weight function $w : E \to \mathbb{R}^+$. Define the **tree sheaf** $\mathcal{T} : G \to$ **Set** for each edge e and each vertex $v \neq v_s$ by

$$\mathcal{T}(v) = \left(\mathrm{In}(v) \times 2^{\mathrm{Out}(v)} \times \mathbb{R}_{\ge 0}\right) \cup \{\bot\}, \text{ and}$$

$$\mathcal{T}(e) = \mathbb{R} \cup \{\bot\}$$

For our source vertex, we set

$$\mathcal{T}(v_S) = 2^{\mathrm{Out}(v_S)},$$

with restriction maps given by

$$\mathcal{T}(v_S \le e)(A) = \begin{cases} 0 & \text{if } e \in A \\ \perp & \text{if } e \notin A, \end{cases}$$

at the source vertex, and

$$\mathcal{T}(v \le e)(e', B, x) = \begin{cases} x - w(e) & \text{if } e \in \mathrm{In}(v) \text{ and } e = e' \\ \perp & \text{if } e \in \mathrm{In}(v) \text{ and } e \ne e' \\ x & \text{if } e \in \mathrm{Out}(v) \text{ and } e \in B \\ \perp & \text{if } e \in \mathrm{Out}(v) \text{ and } e \notin B, \end{cases}$$

with the final case that

$$\mathcal{T}(v \le e)(\perp) = \perp.$$

We can see that sections describe trees as follows. Let σ be a section of $\mathcal{T}$. If $v \ne v_S$ is a vertex such that $\sigma(v) \ne \perp$, then by the definition of the restriction maps, there is exactly one incoming edge e such that $\sigma(e) \ne \perp$. This edge is the first component of $\sigma(v)$ and connects v to its unique parent vertex in the tree. The set of outgoing edges in the second component of $\sigma(v)$ consists of all outgoing edges not assigned to $\perp$ by σ, and these edges connect the vertex to its children. Analogously, for the source, $\sigma(v_S)$ is equal to the set of outgoing edges not assigned to $\perp$ by σ. For any v such that $\sigma(v) \ne \perp$, the final component of $\sigma(v)$ (the numerical value) records the distance from the source: we can move back to each successive parent node and find that these numerical values decrease by edge weights, where the value over each edge agrees with the numerical value over its initial vertex. In particular, we will not find a loop by tracing through successive parents, since the numerical values decrease by the positive edge weights, which implies that the set of vertices and edges not assigned to $\perp$ form a tree. We can always trace back to v_S, so this is a directed tree rooted at v_S. The numerical values stored over the vertices record their distances from v_S, since edges exiting v_S not assigned to $\perp$ are assigned a value of 0.

Since sections of the tree sheaf represent trees rooted at v_S and record of the distance of each vertex from the source, we can reinterpret certain algorithms based on trees in the language of this sheaf. Dijkstra's algorithm and the Bellman–Ford algorithm can both be used to construct shortest-path trees in a connected graph, and they do so by updating a tree consisting of the current best paths. This corresponds to updating a current section of the tree sheaf until an optimal section is found. This illustrates the idea that sheaves can model the data structures used in algorithms, as suggested in [9], [23]. We will explore this idea further in Section 5.

5. Limits of sheaves as gluing of data structures

The primary motivation for incorporating sheaves into networking is the ability of sheaves to describe the gluing of local information into global information. One way of realizing this idea involves using sheaves to record the data structures used in networking algorithms and their relationships. In a sheaf built on a graph, the stalks of the sheaf can record the data type expected to be stored over a vertex or edge during an algorithm, and a section records an instance of each data type. This can model either cases where a graph is an input to an algorithm performed by a single computer or cases where the computation is distributed among multiple computers, represented as vertices in a graph. The restriction maps can then record how these data structures are expected to relate to each other, and how the current data stored forms a section if it is consistent with these expected relationships.

The sheaves we have described up to this point provide examples of storing data on a graph that may be used in an algorithm. In the weights sheaf $\mathcal{W}$ of Section 2, each vertex has knowledge of the weights of its edges, and the restriction maps ensure this record of weights is consistent between vertices. In the tree sheaf $\mathcal{T}$ considered in Section 4, the restriction maps enforce that a section represents a tree. This is accomplished by requiring that each vertex look locally like a tree (that is, it has a unique parent and any number of children) and that the distance from the source is consistently recorded. A similar idea is used to construct the distance path sheaf $\mathcal{DP}$, in which sections represent paths from source to sink and distances from the source are recorded.

In the case of computations distributed among the vertices in the network, the language of sheaves provides a way to describe how decisions made locally at the vertices glue together to form global routing algorithms. This behavior is exemplified in distance–vector routing, where the Bellman–Ford algorithm can be implemented by having each vertex locally update its routing table based on its neighbors, without access to the global structure of the network.

The concept of gluing can happen at a higher level as well: given sheaves on various parts of a network that describe the data structures needed for various algorithms, there is a mathematically optimal way to combine the data structures into a single sheaf. For a simple setting, consider a graph G covered by two open sets U_1 and U_2, where $U_1 \cap U_2$ consists of $\mathrm{star}(v)$ – a single vertex v and all edges incident to it. Suppose we have cellular sheaves of sets $\mathcal{F}_1$ defined on U_1 and $\mathcal{F}_2$ defined on U_2. These tell us the data structures used by algorithms on U_1 and U_2; we would like to construct a sheaf that tells us the appropriate data types to store above all vertices and edges if we are to combine the algorithms. Intuitively, data types in $U_1 \backslash U_2$ and in $U_2 \backslash U_1$ should not need to change, whereas we may expect some process is needed to merge the data structures over $U_1 \cap U_2$. We need v to be able to "speak both languages," in that it must store both the data relevant to U_1 and the data relevant to U_2. The data structures it stores for U_1 and U_2 are $\mathcal{F}_1(v)$ and $\mathcal{F}_2(v)$, so one option would be to require the new data structure over v be the product $\mathcal{F}_1(v) \times \mathcal{F}_2(v)$. However, there might be some overlap in the information captured by $\mathcal{F}_1(v)$ and $\mathcal{F}_2(v)$, and choosing the product may allow components from the two that are not compatible. So instead, we use a structure from category theory for combining potentially overlapping data called a *pullback*.

The pullback $\mathcal{L}(v)$ over v can be defined using the following diagram, where $\mathcal{S}(v)$ and the maps into it record the overlap in information stored by $\mathcal{F}_1(v)$ and $\mathcal{F}_2(v)$.

$$\begin{tikzcd} \mathcal{L}(v) \arrow[r] \arrow[d] & \mathcal{F}_2(v) \arrow[d] \\ \mathcal{F}_1(v) \arrow[r] & \mathcal{S}(v) \end{tikzcd}$$

For a simple example, suppose $\mathcal{F}_1$ is a path sheaf with sink v and $\mathcal{F}_2$ is a path sheaf with source v. Then $\mathcal{F}_1(v)$ is the set of incoming edges to v and $\mathcal{F}_2(v)$ is the set of outgoing edges.

When merging these two algorithms, our interpretation of the sheaves suggests there is no overlap in information being stored by the two sheaves, so in this case, we will let $\mathcal{S}(v) = \{*\}$, a singleton. Then the pullback $\mathcal{L}(v)$ is just a product; the relevant information to store over v is a choice of an incoming edge and an outgoing edge.

However, in general a pullback will be different from a product. For an alternate example, based on a simplified description of a routing table, suppose $\mathcal{F}_1(v) = \mathbb{R}^2$ and $\mathcal{F}_2(v) = \mathbb{R}$, where we interpret a section of $\mathcal{F}_1$ at v as storing the distances to two vertices a and b a section of $\mathcal{F}_2$ at v as storing the distance from the single vertex a. The overlap in the information stored by $\mathcal{F}_1(v)$ and $\mathcal{F}_2(v)$ is the distance to a, so we will let $\mathcal{S}(v) = \mathbb{R}$, where the map $\mathcal{F}_1(v) \to \mathcal{S}(v)$ is the projection onto the first component and the map $\mathcal{F}_2(v) \to \mathcal{S}(v)$ is the identity. In this case, $\mathcal{L}(v) = \mathbb{R}^2$, where the map $\mathcal{L}(v) \to \mathcal{F}_1(v)$ is the identity and the map $\mathcal{L}(v) \to \mathcal{F}_2(v)$ is the projection onto the first component.

So far this has explained what should happen on the stalk of an individual vertex. We now generalize the process somewhat; suppose X is a graph, or more generally a topological space, covered by two open sets U_1 and U_2 and we have sheaves of sets $\mathcal{F}_1$ defined on U_1 and $\mathcal{F}_2$ defined on U_2 (the example above can still be used for visualization). We would like to form the pullback of the two sheaves $\mathcal{F}_1$ and $\mathcal{F}_2$ with maps into a third sheaf $\mathcal{S}$ on $U_1 \cap U_2$ that describes the overlap between $\mathcal{F}_1$ and $\mathcal{F}_2$. However, these sheaves are all defined on different open sets. We can extend each of these sheaves to be defined on all of X by using the *direct image* under inclusion maps. Let $i_1 \colon U_1 \hookrightarrow X$, $i_2 \colon U_2 \hookrightarrow X$, and $i_{1,2} \colon U_1 \cap U_2 \hookrightarrow X$ be inclusion maps. We will write the direct image $(i_1)_*\mathcal{F}_1$ with the shorter notation $\widetilde{\mathcal{F}}_1$, and similarly for i_2 and $i_{1,2}$. For instance, the direct image $\widetilde{\mathcal{F}}_1$ is given by $\widetilde{\mathcal{F}}_1(V) = \mathcal{F}_1(i_1^{-1}(V)) = \mathcal{F}_1(V \cap U_1)$. So we'll now assume we have maps $f_1 \colon \widetilde{\mathcal{F}}_1 \to \widetilde{\mathcal{S}}$ and $f_2 \colon \widetilde{\mathcal{F}}_2 \to \widetilde{\mathcal{S}}$ that record the overlap in information between $\mathcal{F}_1$ and $\mathcal{F}_2$, and form the pullback $\mathcal{L}$ of sheaves.

$$\begin{tikzcd}
\mathcal{L} \arrow[r] \arrow[d] & \widetilde{\mathcal{F}}_2 \arrow[d] \\
\widetilde{\mathcal{F}}_1 \arrow[r] & \widetilde{\mathcal{S}}
\end{tikzcd}$$

Specifically, for an open set V, $\mathcal{L}(V)$ is given by the following pullback diagram.

$$\begin{tikzcd}
\mathcal{L}(V) \arrow[r] \arrow[d] & \widetilde{\mathcal{F}}_2(V) \arrow[d] \\
\widetilde{\mathcal{F}}_1(V) \arrow[r] & \widetilde{\mathcal{S}}(V)
\end{tikzcd}$$

The restriction maps for $\mathcal{L}$ are constructed from the restriction maps of the other sheaves using the universal property of pullbacks, as in the following diagram.

$$\begin{tikzcd}
\mathcal{L}(W) \arrow[rr] \arrow[dd] \arrow[dr, dotted] & & \widetilde{\mathcal{F}}_2(W) \arrow[dr] \arrow[dd] & \\
& \mathcal{L}(V) \arrow[rr] \arrow[dd] & & \widetilde{\mathcal{F}}_2(V) \arrow[dd] \\
\widetilde{\mathcal{F}}_1(W) \arrow[rr] \arrow[dr] & & \widetilde{\mathcal{S}}(W) \arrow[dr] & \\
& \widetilde{\mathcal{F}}_1(V) \arrow[rr] & & \widetilde{\mathcal{S}}(V)
\end{tikzcd}$$

For any $V \subseteq U_1 \setminus U_2$, we have $\widetilde{\mathcal{F}}_2(V) = \mathcal{F}_2(V \cap U_2) = \mathcal{F}_2(\varnothing) = \{*\}$, where $\{*\}$ is any singleton, and similarly $\widetilde{\mathcal{S}}(V) = \{*\}$. We also have $\widetilde{\mathcal{F}}_1(V) = \mathcal{F}_1(V \cap U_1) = \mathcal{F}_1(V)$, so computing $\mathcal{L}(V)$ from the pullback diagram above, we find $\mathcal{L}(V) \cong \mathcal{F}_1(V)$. For any $v \in V$, this shows $\mathcal{L}_v \cong \left(\widetilde{\mathcal{F}}_1\right)_v$. Therefore, $\mathcal{L}$ does not change the value on open sets or stalks in the interior of $U_1 \setminus U_2$ or $U_2 \setminus U_1$, so we interpret this as keeping the same data types outside of the intersection.

The universal property of pullbacks provides justification that this is the "correct" way to combine data types described by $\mathcal{F}_1$ and $\mathcal{F}_2$. Suppose we are given a sheaf $\mathcal{G}$ such that the following diagram commutes.

$$\begin{tikzcd}
\mathcal{G} \arrow[r] \arrow[d] & \widetilde{\mathcal{F}}_2 \arrow[d] \\
\widetilde{\mathcal{F}}_1 \arrow[r] & \widetilde{\mathcal{S}}
\end{tikzcd}$$

In our interpretation, $\mathcal{G}$ records data types that are able to recover the data types of $\mathcal{F}_1$ and $\mathcal{F}_2$, consistent with how they overlap, as described by $\mathcal{S}$. The universal property of pullbacks states that there exists a unique map $\mathcal{G} \to \mathcal{L}$ such that the resulting diagram commutes. That is, if $\mathcal{G}$ is able to describe the data types of $\mathcal{F}_1$ and $\mathcal{F}_2$, then it must require us to store at least as much information as $\mathcal{L}$.

We can now provide more details on the first simple example. Let G be a directed graph covered by two open sets U_1 and U_2, where $U_1 \cap U_2$ consists of a single vertex v_0 and all edges connected to it. Suppose $s \neq v_0$ is a vertex in U_1 and $t \neq v_0$ is a vertex in U_2. We define path sheaves on these open sets. Define $\mathcal{F}_1$ on U_1 as follows.

$$\mathcal{F}_1(s) = \mathrm{Out}(s)$$

Let $\mathcal{F}_1(v_0) \subseteq \mathrm{In}(v_0)$ be the set of incoming edges of v_0 whose other vertices are contained U_1.

For vertices v not equal to s or v_0,

$$\mathcal{F}_1(v) = (\mathrm{In}(v) \times \mathrm{Out}(v)) \cup \{\perp\}.$$

If both vertices of an edge e in U_1 are contained in U_1,

$$\mathcal{F}_1(e) = \{\top, \perp\}.$$

If one vertex of an edge e in U_1 is not in U_1 (note this can only be true for edges incident to v_0), then

$$\mathcal{F}_1(e) = \{*\}.$$

For restriction maps, if a vertex v is not equal to s or v_0, then for any edge e connected to v, let

$$\mathcal{F}_1(v \leq e)(e_1, e_2) = \begin{cases} \top & \text{if } e = e_1 \text{ or } e = e_2 \\ \bot & \text{otherwise,} \end{cases}$$

$$\mathcal{F}_1(v \leq e)(\bot) = \bot.$$

For any edge e connected to s, let

$$\mathcal{F}_1(s \leq e)(e') = \begin{cases} \top & \text{if } e = e' \\ \bot & \text{in } e \neq e'. \end{cases}.$$

For any edge e connected to v_0 whose other vertex is contained in U_1, let

$$\mathcal{F}_1(v_0 \leq e)(e') = \begin{cases} \top & \text{if } e = e' \\ \bot & \text{in } e \neq e'. \end{cases}.$$

And finally if e is an edge connected to v_0 whose other vertex is not contained in U_1, let

$$\mathcal{F}_1(v_0 \leq e)(e') = *.$$

Define $\mathcal{F}_2$ on U_2 analogously, with $\mathcal{F}_2(t) = \mathrm{In}(t)$ and $\mathcal{F}_2(v_0)$ equal to the set of outgoing edges of v_0 whose other vertices are contained in U_2. The appropriate choice of a sheaf $\mathcal{S}$ on $U_1 \cap U_2$ depends on our interpretation of the other sheaves. In this case, we expect to use sections of the path sheaves to describe paths. The value of a section of $\mathcal{F}_1$ at v_0 records an incoming edge, determining the end of the path through U_1. The value of a section of $\mathcal{F}_2$ at v_0 records an outgoing edge, determining the beginning of the path through U_2. Thus, there is no expected overlapping information stored by the sections of these two sheaves, so we will let the stalks of $\mathcal{S}$ be a singleton $\{*\}$ over v_0 and over each edge.

We can form the pullback $\mathcal{L}$, a sheaf on $U_1 \cup U_2$ as described above. Since $\mathcal{S}(v_0) = \{*\}$, the stalk $\mathcal{L}(v_0)$ is given by the product $\mathcal{F}_1(v_0) \times \mathcal{F}_2(v_0)$. That is, each element is a pair consisting of an incoming edge with vertices in U_1 and an outgoing edge with vertices in U_2. Stalks over edges connected to v_0 are products as well. However, if an edge connected to v_0 has its other vertex in U_1, then $\mathcal{F}_1(e) = \{\top, \bot\}$ while $\mathcal{F}_2(e) = \{*\}$, so we can identify the product with $\{\top, \bot\}$. The same goes for edges connected to v_0 with their other vertex in U_2. Working through the definition of the restriction maps for $\mathcal{L}$ from v_0 to edges, we get the intuitive result that

$$\mathcal{L}(v_0 \leq e)(e_1, e_2) = \begin{cases} \top & \text{if } e = e_1 \text{ or } e = e_2 \\ \bot & \text{otherwise.} \end{cases}$$

Other restriction maps can be checked to give the expected results as well. We thus find that $\mathcal{L}$ nearly matches the original definition of a path sheaf on $U_1 \cup U_2$ with source s and sink t with only a modification to the stalk above v_0, requiring incoming edges have their other vertices in U_1 and outgoing edges have their other vertices in U_2. This matches the intuition of gluing together a path from s to v_0 and a path from v_0 to t to form a path from s to t.

So far, we have only considered the case of two open sets U_1 and U_2 that cover a space, where we have formed a pullback from sheaves on these open sets mapping to a sheaf on the intersection $U_1 \cap U_2$. We can generalize this to larger open covers by using more general category theoretic *limits*, of which a pullback is an example. The construction is similar to what we have described with pullbacks, involving sheaves that record overlapping information on all intersections and extending each sheaf using direct images. Limits come with similar universal properties, justifying them as the proper way to combine data types that are stored by sheaves.

6. Distributed routing via products of sheaves

In the previous section, we saw that combinations of multiple sheaves may be used to represent heterogenous data structures over the same topological space. In this section, we will show that similar ideas can be used to model distributed problems over a network. For concreteness, we will focus our discussion on the following specific distributed routing problems.

Problem 6.1. *(k-Consistent Routing) Suppose we have a graph G with a fixed sink node O, which we refer to as the origin. Suppose each vertex in G has access to the list of possible paths it may use to send messages to O, and furthermore has ranked these routes in order of preference. If a packet is sent to O from some vertex v, is there a routing protocol such that the packet can be sent along a route that is highest ranked by all vertices on this route? More generally, for $k \in \mathbb{N}$, is there a routing protocol such that the packet can be sent along a route that is in the list of k most highly ranked routes for each vertex on the route?*

Problem 6.2. *(k-Quasi-Consistent Routing) Suppose each vertex in G sends packets according to their preferred routes. Is every vertex eventually able to reach the origin O? More generally, for $k \in \mathbb{N}$, if every vertex sends packets according to one of their k most highly ranked routes, can every vertex reach the origin O?*

Clearly, the solution to the k-Consistent Routing problem provides a solution to the k-Quasi-Consistent Routing problem. Notice that both problems depend on a chosen ranking on the set of paths from each vertex to O. We will now show how objects we call *ranked path sheaves* can encode these two problems, and that obstructions to finding solutions can be detected by topology.

First, we present a slight modification of the distance path sheaf (described in Section 3). This sheaf, $\mathcal{P}_v$, will have global sections representing directed paths from source v to sink O. Before providing the formal definition of $\mathcal{P}_v$, it will be helpful to understand how local sections $\mathcal{P}_v(w)$, $w \in V$ and $\mathcal{P}_v(e)$, $e \in E$ assemble to a global section. We let $\Gamma(\mathcal{P}_v)$ be the set of global sections of $\mathcal{P}_v$. In order to describe a directed path from v to O, every vertex $w \neq v \in V$ must be assigned sufficient data to determine:

- Is w in the path?
- If so, along which edges does the path travel?
- What is the direction the path takes through these edges?
- How far along the path is w?

Informally, the first query can be answered with a binary "on or off" component, whereas the second and third queries can be answered by a pair of edges, one labelled "in" and the other labelled "out". For this, the relevant data can be described as follows: Let $E(v)$ be the set of edges incident to v. Then let $H(v) = E(v) \times E(v) \setminus \{(e_i, e_i) | e_i \in E(v)\}$. For convenience, we will sometimes refer to the first two factors in $H(v)$ as $E^{in}(v)$ and $E^{out}(v)$. Notice that by removing the set $\{(e_i, e_i) | e_i \in E(v)\}$ we have prevented a path from being both “in” and “out”. The final piece of information can be given by a nonnegative integer and is not strictly necessary to define a path to O. However, as discussed in [23], removing this piece of data results in introducing additional global sections of $\mathcal{P}_v$ that are not directed paths. For source v and origin O, similar data is required, though both need to store only one copy of $E(v)$, representing E^{out} and E^{in} respectively.

For an edge $e \in E$, it must be assigned sufficient data to determine:

- Is e in the path?
- If so, what is the direction the path takes through e?
- How long is e?

The above queries can be answered with an “on” or “off” component, an orientation, and a nonnegative integer. The orientation can be stored by indicating which of the edge’s boundary vertices is the path traveling “in” from. As before, the length of e is included for technical reasons. Given the vertex and edge data, the restriction maps are chosen so that sections correctly glue into a path, which means:

- The “out” edge of one vertex is the “in” edge of a neighboring vertex;
- An “off” vertex is not incident to “on” edges;
- v (resp. O) is the first (resp. last) vertex in the path;
- The length of the path is the sum of the length of its edges.

This is sufficient information to define the sheaf $\mathcal{P}_v$:

Definition 6.3. Let $G = (V, E)$ be a digraph with identified source vertex $v \in V$ and origin O. Then, the **directed path sheaf at $\boldsymbol{v}$**, denoted $\mathcal{P}_v$ is defined at each vertex by

$$\mathcal{P}_v(w) = \begin{cases} E(v) \times \{0\} & \text{if } w = v \\ E(v) \times \mathbb{N} & \text{if } v = O \\ (H(v) \times \mathbb{N}) \cup \{\perp\} & \text{otherwise.} \end{cases}.$$

For edge e with boundary vertices u, w,

$$\mathcal{P}_v(e) = (\{u, w\} \times \mathbb{N}) \cup \{\perp\}.$$

Then, the restriction maps for edges e incident to v and a vertex w are given by

$$\mathcal{P}_v(v \leq e)(e_i, 0) = \begin{cases} \{v\} \times 1 & \text{if } e = e_i \\ \perp & \text{otherwise} \end{cases}.$$

Also, the restriction maps for edges e incident to O and a vertex w are given by

$$\mathcal{P}_v(O \leq e)(e_i, x) = \begin{cases} \{w\} \times x & \text{if } e = e_i \\ \perp & \text{otherwise} \end{cases}.$$

If w is a non-sink and a non-source node with non-$\perp$ assignment, define

$$\mathcal{P}_v(w \leq e)(e_i, e_j, x) = \begin{cases} x & \text{if } e = e_i \\ x + w(e) & \text{if } e = e_j \\ \perp & \text{otherwise} \end{cases}.$$

Then, finally, if $\mathcal{P}_v(w) = \perp$, define

$$\mathcal{P}(w \leq e)(\perp) = \perp.$$

The following proposition is key to why we call this the directed path sheaf starting at v.

Proposition 6.4. *Global sections of $\mathcal{P}_v$ define directed paths from v to O.*

Proof: An analogous proof exists in [23]. □

Note that the condition on the restriction maps prevents two adjacent vertices from having the same ”on” edge, as this would correspond to an invalid section. Denote global sections of $\mathcal{P}_v$ by $\Gamma(\mathcal{P}_v)$, which correspond to directed paths from v to O. Let $p_v = |\Gamma(P_v)|$, i.e. the number of such paths.

Definition 6.5. We call a bijection

$$r_v : \Gamma(\mathcal{P}_v) \to \{1, 2, \ldots, p_v\}$$

a **path ranking** at v. We call a path sheaf equipped with a path ranking a **ranked path sheaf**.

As a data structure, the ranked path sheaf at v records the set of possible routes from v to the origin O as well as a preference ranking on this set of routes $\Gamma(P_v)$. We will fix the ranking so that $r_v(\sigma) = 1$ denotes the most favored section for $\mathcal{P}_v$. We need one more construction in order to begin modeling the problems of interest, which is the product of sheaves. The definition of a product of sheaves appears implicitly in the above section as a special case of the limit of sheaves construction. But the idea can be explained without reference to limits: given a pair of sheaves $\mathcal{F}_\infty, \mathcal{F}_\in$ on a graph G, which for instance have sections corresponding to two different data structures on G, the product $\mathcal{F}_\infty \times \mathcal{F}_\in$ has sections given by a pair of sections, one from $\mathcal{F}_\infty$ and the other from $\mathcal{F}_\in$. We now present a more formal description that is sufficient for our application:

Definition 6.6. Let $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_m$ be (set-valued) cellular sheaves on a graph G. Then the **product sheaf** $\prod_{i=1}^{m} \mathcal{P}_i = \mathcal{P}_1 \times \mathcal{P}_2 \times \ldots \mathcal{P}_m$ is the cellular sheaf with stalks on vertices $v \in G$ given by

$$\Big[\prod_{i=1}^{m} \mathcal{P}_i\Big](v) = \mathcal{P}_1(\sigma) \times \ldots \times \mathcal{P}_m(v)$$

where × denotes the Cartesian product of sets, and restriction maps from $v \leq e$ are defined as the natural Cartesian product of set valued maps

$$\Big[\prod_{i=1} \mathcal{P}_i\Big](v \leq e) = \mathcal{P}_1(v \leq e) \times \ldots \mathcal{P}_m(v \leq e).$$

As mentioned, the set of sections of a product of sheaves is given by the product of the sections of its *factors*, i.e. $\Gamma\Big(\prod_{i=1}^{m} \mathcal{P}_i\Big) = \prod_{i=1}^{m} \Gamma(\mathcal{P}_i)$. Now consider the sheaf $\prod_{v \in V}^{m} \mathcal{P}_v$. This is a product of path sheaves, one for each $v \in V$. The global sections of this sheaf are given by a choice of path from each $v \in V$ to O, which we may record as $(\sigma_1, \sigma_2, \ldots, \sigma_m)$ with $\sigma_i \in \Gamma(\mathcal{P}_i)$ (where we have enumerated $v \in V$ with numbers from 1 to $m = |V|$). By the product construction, these paths are independent of one another, in the sense that for any $v \neq w$ the choice of section at $\mathcal{P}_v$ does not inform us about the section at $\mathcal{P}_w$.

Now suppose that each directed path sheaf $\mathcal{P}_v$ is equipped with a ranking r_v. We may use these rankings to define a ranking on the set $\Gamma\Big(\prod_{i=1}^{m}\mathcal{P}_i\Big)$. There are numerous ways one may combine multiple rankings on a collection of sets into a ranking on the product of such sets. For instance, one may assign to each product the *average* rank of its factors:

Definition 6.7. For each $v \in V$, let r_v be a ranking on $\Gamma(\mathcal{P}_v)$. Then the **sum-rank** r_1 is defined $r_1(\sigma_1, \sigma_2, \ldots, \sigma_m) = \frac{1}{m}\sum_{i=1}^{n} r_i(\sigma_i)$ for all $(\sigma_1, \ldots, \sigma_m) \in \Gamma\Big(\prod_{i=1}^{m}\mathcal{P}_i\Big)$.

To describe consistent and quasi-consistent routing, we will require a different ranking, i.e. the one that assigns to each $(\sigma_1, \sigma_2, \ldots, \sigma_m)$ the rank of its "worst" factor:

Definition 6.8. For each $v \in V$, let r_v be a ranking on $\Gamma(\mathcal{P}_v)$. Then the **sup-rank** r_∞ is defined $r_\infty(\sigma_1, \sigma_2, \ldots, \sigma_m) = \max_i r_i(\sigma_i)$ for all $(\sigma_1, \ldots, \sigma_m) \in \Gamma\Big(\prod_{i=1}^{m}\mathcal{P}_i\Big)$.

We now have the requisite terminology to define k-consistent and k-quasi-consistent routing plans. In particular, these plans are given by distinguished subsets of the set of all sections of $\Gamma\Big(\prod_{i=1}^{m}\mathcal{P}_i\Big)$.

Definition 6.9. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) \in \prod_{v\in V}\mathcal{P}_v$. Then σ is **consistent** if for all $v, w \in V$ such that $\sigma_v(w) \neq \perp$, then $\sigma_v(\sigma) \neq \perp$ for all simplices $\sigma \in G$ such that $\sigma_w(\sigma) \neq \perp$. In other words, if w is on the path from v to O defined by σ_v, then σ_w is a subpath of σ_v.

Quasi-consistent routing plans will also correspond to a distinguished subset of $\Gamma\Big(\prod_{i=1}^{m}\mathcal{P}_i\Big)$ (which will, in fact, contain the set of consistent sections). To describe this set we will need additional terminology:

Let $v \in V$ and recall that the *star* at v is the set containing v and each edge incident to v and is denoted $\mathrm{star}(v)$ We need to restrict r_v from a ranking on $\Gamma(\mathcal{P}_v)$ to a ranking r_v^* on $\Gamma(\mathcal{P}_v|_{\mathrm{star}(v)})$. We do this by assigning each restricted section $\sigma_v|_{\mathrm{star}(v)}$ the maximum rank of any section σ_v' such that $\sigma_v|_{\mathrm{star}(v)} = \sigma_v'|_{\mathrm{star}(v)}$, i.e. r_v^* assigns, to each local section $\sigma|_{\mathrm{star}(v)}$, the "best" rank of any global section that agrees with $\sigma|_{\mathrm{star}(v)}$ on the neighborhood of v. In other words, for all $\sigma_v|_{\mathrm{star}(v)} \in \Gamma(\mathcal{P}_v|_{\mathrm{star}(v)})$:

$$r_v^*(\sigma_v|_{\mathrm{star}(v)}) = \max(r_v(\sigma_v'|_{\mathrm{star}(v)})) : \sigma_v'|_{\mathrm{star}(v)} = \sigma_v|_{\mathrm{star}(v)}).$$

We call this *the ranking induced by* r_v *on* $\mathrm{star}(v)$. Notice that by definition of $\mathcal{P}_v$, for all $\sigma_v \in \Gamma(\mathcal{P}_v)$, we can describe $\sigma_v|_{\mathrm{star}(v)}$ as follows:

$$\sigma_v|_{\mathrm{star}(v)}(v) = e$$

$$\sigma_v|_{\mathrm{star}(v)}(e) = (v, w)$$

$$\sigma_v|_{\mathrm{star}(v)}(e') = \perp, \forall e' \neq e$$

for some choice of $e \in \mathrm{star}(v)$ with boundary vertices v and w.

Definition 6.10. Let $\sigma_v \in \Gamma(\mathcal{P}_v)$, and let $U \subset G$. Then $\overset{\circ}{\sigma}_v|_U$ is the set of all stalks $\sigma_v(g)$ for $g \in U$ (where g is a vertex or an edge) such that $\sigma_v(g) \neq \perp$. One may informally think about $\overset{\circ}{\sigma}_v|_U$ as the set of vertices and edges in the set U which are "on" in σ.

Definition 6.11. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) \in \prod_{v\in V}\mathcal{P}_v$. We have that σ is **quasi-consistent** if for all $v, w \in V$ such that $\sigma_v(w) \neq \perp$, then $\overset{\circ}{\sigma}_w|_{\mathrm{star}(w)} \hookrightarrow \overset{\circ}{\sigma}_v|_{\mathrm{star}(w)}$.

Theorem 6.12. *Let G be a graph and let* $\prod_{v\in V}\mathcal{P}_v$ *be a product of path sheaves* $\mathcal{P}_v$ *with source node* $v \in V$ *and sink node* O*. Let* $\{r_v\}_{v\in V}$ *be a set of path rankings at each vertex and let* r_∞ *be the sup-rank from this set. Let* $\sigma \in \Gamma(\mathcal{P}_v)$*. Then:*

- *If* σ *is consistent and* $r_\infty(\sigma) \geq k$*, then* σ *determines a k-consistent routing plan.*
- *If* σ *is quasi-consistent and* $r_\infty(\sigma) \geq k$*, then* σ *determines a k-quasi-consistent routing plan.*

The proof follows by inspecting the various definitions in the statement, and hence we leave it as an instructive exercise.

The above theorem shows that ranked path sheaves are capable of modeling natural distributed routing problems on graphs, where sections of a product of ranked path sheaves correspond to particular routing plans. Notice that consistent sections are also quasi-consistent, though the converse is certainly not true. The existence of a k-(quasi-)consistent section is not guaranteed, as there will be many choices of path rankings that contradict one another. It is an interesting question on how to efficiently detect and resolve such obstructions. In particular, one may ask how many path rankings r_v must be replaced before a single (quasi)-consistent section exists. Given a set of path rankings, it is not hard to show that this number can be bounded below using topological methods:

Definition 6.13. Let $\sigma \in \Gamma\Big(\prod_{v\in V}\mathcal{P}_v\Big)$. Then the **support** of σ, $\mu(\sigma) \subset G$ is the subgraph given by the union of cells g such that $\sigma_v(g) \neq \perp$ for some $v \in V$.

Proposition 6.14. *Let* $\sigma \in \Gamma\Big(\prod_{v\in V}\mathcal{P}_v\Big)$*. For any subgraph* $H \subset G$*, let* $\beta_0(S)$ *denote the number of connected components of* H*, and* $\beta_1(S)$ *the number of cycles in* H*.* $\max\{\beta_0(\mu(\sigma)) - 1, \beta_1(\mu(\sigma))\}$ *is a lower bound on the number of path rankings that must be changed before* σ *may be extended to a global quasi-consistent section.*

Proof: To prove this we will first characterize how cycles and disconnected components in $\mu(\sigma)$ may arise from a partial section σ:

Suppose there is a cycle $C \in \mu(\sigma)$. In order to extend σ to a quasi-consistent section, we must be able to identify an edge in $\mathrm{star}(v)$ as the "out" edge for $\sigma_v(v)$ for each $v \in V$. Each vertex in C has two incident edges that are also part of C. Let $v, v_1, v_2 \in C$ such that v is connected to v_1 by e_1 and v_2 by e_2. Clearly, e_1, e_2 cannot both be "out" edges of v, as this would result in a not well-defined stalk $\sigma_v(v)$. Suppose that $\sigma_{v_1}(e_1) = v_1$, $\sigma_{v_2}(e_2) = v_2$, i.e. both edges e_1 and e_2 are pointing "in" to v_1. It is not hard to show (for instance, by induction) that this implies that there must be a vertex w in the cycle such that w has two "out" edges, and hence $\sigma_w(w)$ will not be well defined. Thus we have shown that if a cycle C appears in $\mu(\sigma)$, then each $v \in C$ with adjacent edges $e_1, e_2 \in C$ cannot have both edges pointing "in" to v, and hence $\sigma_v(v) = e_1$ or $\sigma_v(v) = e_2$. Notice that this includes any case where $O \in C$, as by definition both edges incident to O in C must be "in" edges.

Now suppose there is a component D disconnected from O in $\mu(\sigma)$. There is some set of vertices $w_1, w_2, \ldots, w_m$ in D such that for each w_i there is no incident edge e with boundary v, w_i such that $\sigma_v(v) = e$, i.e. the w_i are the vertices of D with no "in" edges. Since D is not connected to O, if we follow the sequence γ_i of "out" edges defined by σ starting at any w_i, we see that eventually there must be a vertex v_i in γ_i such that $\sigma_{v_i}(v_i) = q_i$ for some $q_i \in D$ already visited by γ_i. If q_i is the vertex directly preceding v_i in γ_i, then v_i is either part of a cycle in D or is a leaf (valence 1 vertex) of D. If q_i is not directly preceding v_i in γ_i, then this creates a cycle.

In all cases, we see that each of the above v_i in γ_i creates an obstruction to extending σ to a quasi-coherent section. In particular, $\sigma_{v_i} \in \Gamma(\mathcal{P}_{v_i})|_{\mathrm{star}(v_i)}$ must be changed so that $\sigma_{v_i}(v_i)$ does not intersect with γ_i. Since each v_i is either part of a cycle or a leaf node on a disconnected component, the number of such v_i is bounded below by $\max\{\beta_0(\mu(\sigma)) - 1, \beta_1(\mu(\sigma))\}$. A better lower bound based solely on β_0 or β_1 does not exist: We may have a disconnected component with many cycles and no leaves, or a disconnected component with no cycles and many leaves. □

Corollary 6.15. *If a section* $\sigma \in \Gamma\Big(\prod_{v\in V} \mathcal{P}_v\Big)$ *is consistent (quasi-consistent) then* $\mu(\sigma)$ *is a tree, i.e. a connected graph with no cycles.* □

Remark 6.16. There exists a more elegant proof of the above theorem using discrete Morse theory.

7. Zigzag Persistence and the Edges Cosheaf

Ultimately, we wish to know as much as possible about the structure of a network in order to make the best routing decisions possible. The structure we care about is topological in nature, and so we employ techniques from computational topology to capture this. The two primary tools of use here are persistent homology and *zigzag persistence* [25]. The typical output of both of these methods is the *barcode*.

Standard persistent homology operates by toggling a single parameter, such as scale, from lower values to higher values and then organizes this into a *filtration*. In this setting, the barcode captures at what scales topological features are born and at what scales they die. Zigzag persistence is strictly more general and allows us to add or delete nodes and edges as a function of time.

It is typical to cast zigzag persistence in terms of a *two-parameter* filtration, the two parameters being the start time and end time of a window in time. One then captures the births and deaths of features by letting these endpoints vary and computing changes of homology along each of these filtrations.

Although there are other methods that model persistence of weighted directed graphs [26], our approach considers a dynamic digraph (e.g., temporal network, directed), from this two-parameter perspective. Although typically two-parameter persistence is considered computationally unwieldy, our setup leverages the connection with zigzag persistence to ensure computability. This connection is formalized via cosheaves, which we explore in this section. We begin by defining *dynamic digraphs*, which model networks as they evolve over time.

Definition 7.1 ([27])**.** A **digraph** $G = (V, E)$ is a set of vertices together with a set of edges $E \subseteq V \times V \setminus \mathrm{diag}(V)$. A morphism of digraphs $G = (V_G, E_G) \to H = (V_H, E_H)$ is a map $f : V_G \to V_H$ such that if $e = (u, v) \in E_G$ then $f(e) = \big(f(u), f(v)\big)$ is either an edge in H or $f(u) = f(v)$ in which case $f(e)$ collapses (as no loops are allowed). We denote the category of digraphs by $\mathsf{Digraph}$.

Remark 7.2. It can be shown that this category of digraphs admits (small) limits and colimits.

The zigzag poset $\mathbf{ZZ}$ is the subposet of $\mathbb{R}^{\mathrm{op}} \times \mathbb{R}$ given by $\mathbf{ZZ} := \big\{(\mathbf{i}, \mathbf{j}) : \mathbf{i} \in \mathbb{Z}, \mathbf{j} \in \{\mathbf{i}, \mathbf{i-1}\}\big\}$. We write $i = (i, i)$ and $\underline{i} := (i, i-1)$ for simplicity.

$$\begin{tikzcd}
 & & i+1 \\
 & i & \underline{i+1} \arrow[l] \arrow[u] \\
i-1 & \underline{i} \arrow[l] \arrow[u] &
\end{tikzcd}$$

Definition 7.3. A **dynamic digraph** (G, w_G) consists of a functor $G : \mathbf{ZZ} \to \mathsf{Digraph}$ together with an monotone increasing map $w_G : \mathbb{Z} \to \mathbb{R}$, called the **temporal weight**, which is used to track the *critical times* when the underlying digraph changes. A morphism of dynamic digraphs $G \to H$ consists of two natural transformations, $(G, w_G) \Rightarrow (H, w_H) : \mathbf{ZZ} \to \mathsf{Digraph}$ and $w_G \Rightarrow w_H : \mathbb{Z} \to \mathbb{R}$. These form a category of dynamic digraphs, denoted by $\mathsf{DynDigraph}$.

Given a dynamic digraph (G, w_G) on n-vertices (that is, $|V| = |V_{G(k)}| = n$, for all $k \in \mathbb{Z}$), there is a associated **dynamic digraph bifiltration**, $G^{\updownarrow} : \mathbb{R}^{\mathrm{op}} \times \mathbb{R} \to \mathsf{Digraph}$, defined by $G^{\updownarrow}(a, b) = \big(V, E^{\updownarrow}(a, b)\big)$ with $(u, v) \in E^{\updownarrow}(a, b)$ if $(u, v) \in E_{G(k)}$ for some $k \in \mathbb{Z}$ with $w_G(k) \in (a, b)$. This can be thought of as a type of interlevel digraph bifiltration arising from the temporal weight.

To summarize the topology of this bifiltration, we apply the **path homology functor** (as described in [27]) to it, which yields a functor $\mathbf{ZZ} \to \mathbf{Vect}$, alternatively called a **zigzag module**. Zigzag modules are endowed with a natural extended pseudometric known as the **interleaving distance**. This distance is effectively computable, by virtue of the isometry theorem of [28, Proposition 4.18, p. 115], which relates the interleaving distance to the bottleneck distance, which can be computed in polynomial time. These results, and their technical requirements, are summarized in the next result.

Lemma 7.4. *Given a (pointwise) finite dynamic digraph (G, w_G), applying the path homology functor to $G^{\updownarrow}$ yields a block decomposable functor which means that the barcode distance coincides with the interleaving distance, that is, for G, H dynamic digraphs,*

$$d_{\mathsf{B}}\big(\mathcal{B}\mathbf{pH}_k(G^{\updownarrow}), \mathcal{B}\mathbf{pH}_k(H^{\updownarrow})\big) = d_{\mathsf{I}}\big(\mathbf{pH}_k(G^{\updownarrow}), \mathbf{pH}_k(H^{\updownarrow})\big).$$

Proof. This follows from the fact that the k-dimensional path homology of a finite digraph is finite-dimensional. Every pointwise finite-dimensional zigzag module is block decomposable by [29, Proposition 4.2, p. 15]. By [28, Proposition 4.18, p. 115], the interleaving distance and the barcode distance on the zigzag modules are equal. □

Remark 7.5. Applying simplicial homology to the underlying dynamic graph yields the analogous result

$$d_{\mathsf{B}}\big(\mathcal{B}\mathbf{H}_k(G^{\updownarrow}), \mathcal{B}\mathbf{H}_k(H^{\updownarrow})\big) = d_{\mathsf{I}}\big(\mathbf{H}_k(G^{\updownarrow}), \mathbf{H}_k(H^{\updownarrow})\big).$$

We note that homological degree-0 barcodes for zigzag homology (of the underlying undirected dynamic graph) and zigzag path homology coincide and correspond to the barcode of the Reeb graph associated to the dynamic digraph.

Lemma 7.6. *Given a dynamic digraph (G, w_G), then zero-dimensional zigzag path homology corresponds with the zero-dimensional zigzag simplicial homology of the underlying undirected dynamic graph, which tracks the connected components through time.*

Remark 7.7. Unlike the undirected case, the p-th zigzag path homology of a dynamic digraph may be nonzero, possibly encoding useful information of the network.

The dynamic digraph bifiltration can be interpreted as a Digraph-valued cosheaf on $\mathbb{R}$, called the **edges cosheaf**. Note that this is not a cosheaf on a poset, but the more general notion of a cosheaf over a space, in this case over $\mathbb{R}$ as studied in [30]. In particular, the cosheaf interleaving distance coincides with the zigzag interleaving distance [29, Remark 4.12].

Lemma 7.8. *If (G, w_G) is a dynamic digraph then the functor $G : \mathbb{R}^{op} \times \mathbb{R} \to \mathsf{Digraph}$ defined by*

$$(a, b) \mapsto \big\{\textit{edges alive in time interval } (a, b)\big\} = G^{\updownarrow}(a, b),$$

is a cosheaf on $\mathbb{R}$.

Proof. Let $I = (a, b)$ be an open interval and let $\mathcal{U}_I := \{I_r\}$ be an open cover of I by connected open intervals. We want to show that $G(I)$ satisfies the coequalizer condition,

$$\begin{tikzcd}
\bigsqcup_{p,q} G(I_{p,q}) \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & \bigsqcup_{r} G(I_r) \arrow[r, "\eta"] \arrow[dr, "\varepsilon"'] & G(I) \arrow[d, dashed, "\exists!\underline{\varepsilon}"] \\
 & & X
\end{tikzcd}$$

given $\varepsilon : \sqcup_r G(I_r) \to X$ with $\varepsilon f = \varepsilon g$, there exists $\underline{\varepsilon} : G(I) \to X$ such that $\underline{\varepsilon} \circ \eta = \varepsilon$.

If $e \in G(I)$ then it exists for some open interval J in I. This interval J must intersect some I_r nontrivially. Define $\underline{\varepsilon}(e) := \varepsilon_r(e)$. We must show that this is well-defined – if e exists during intervals I_p and I_q, then since $\varepsilon f = \varepsilon g$, we have that $\varepsilon f_{pq}(e) = \varepsilon g_{pq}(e)$, which is equivalent to $\varepsilon_p(e) = \varepsilon_q(e)$, as desired. To see that $\underline{\varepsilon}$ is unique, suppose some $\gamma : G(I) \to X$ has the property that $\gamma\eta = \varepsilon$. Then for $e \in G(I_p)$, we have $\gamma\eta(e) = \gamma(e) = \varepsilon_p(e)$, establishing that $\underline{\varepsilon} = \gamma$. □

To be precise, this is the definition of a *cosheaf on the basis* of bounded open intervals of $\mathbb{R}$. That this extends to a legitimate cosheaf on $\mathbb{R}$ follows from taking the left Kan extension of this along the inclusion of the basis to the topology on $\mathbb{R}$; this can be done since Digraph has small colimits.

Given the notion of a *weighted* dynamic digraph, we can associate a few three-parameter filtrations, for example, two parameters could filter time, focusing on a bounded interval (a, b) as above, and one parameter could track a sublevelset filtration on the edge weights. This is an avenue of future investigation as the computational aspects of three-parameter modules is not yet well developed.

8. Gradient Descent for Network Coding and Routing Decisions

To this point, our focus has been on the construction of sheaves for determining routing structures. However, the purpose of these structures is to enable us to use the information provided to analyze network capabilities. In this section, we outline a method for using machine learning techniques to make routing decisions in a network. Specifically, we present a way to use gradient descent to make network coding decisions. Instead of developing new sheaves, we rely on the multicast sheaf structures from [9], [24] to provide the organizational structure for our analysis.

A key step in the analysis involves finding sections of a cellular sheaf as discussed in Section 2. The most straightforward approach to finding sections of a cellular sheaf of sets is a challenging combinatorial problem with exponential complexity. In this section, we show a way to reduce the combinatorial approach to one that uses linear algebra and gradient descent. This reduces the problem of finding sections to one with polynomial complexity.

Network coding is a useful tool for improving the capacity utilization of a network, without changing the structure of the network itself [31]. However, deciding how to implement network coding on a general network tends to be a very difficult problem. How network coding is implemented depends crucially on the structure of the network itself. Thus, finding a general method for implementing network coding that works on any network is a challenging problem.

The key technical reason networking coding is difficult is that we need to make local decisions at each vertex of the graph in such a way that they all combine to produce the desired result. Cellular sheaves are well-suited to handle these complex local-to-global relationships in a tractable way. In the context of a network, cellular sheaves describe how the local flow of data through a network relates to the global flow of information through the network.

Networking Sheaves

We will introduce multicast indexing sheaves in this section. A given multicast indexing sheaf can be thought of as the rules for how the vertices of our network are allowed to communicate with each other. For example, the unicast indexing sheaf only allows a vertex to communicate with itself and with one vertex of a different name at a given time. A *section* of a multicast indexing sheaf represents a specific way to route data through our network, subject to the rules of the sheaf.

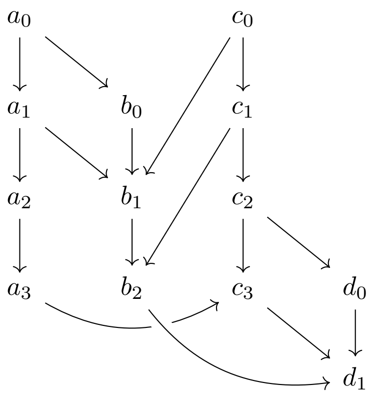
We care about the sections of multicast indexing sheaves because they provide a useful way to describe possible routes through the network. We describe in Section 8 how to use the sections that we find to implement gradient descent to make network coding decisions.

Before we can talk in detail about the multicast indexing sheaves, we will need to describe the graphs that they live over. These are simple directed graphs that keep track of different objects over different time steps. Specifically, each vertex of the graph represents an object at a given time step. The advantage of these *time-extended graphs* is that they can handle many of the challenges faced in space networks, such as relativistic effects, signal latency, changes in the network over time, and a lack of end-to-end connections at any given time.

Time-Extended Graphs—For this section, we assume that our directed graphs are **simple**, i.e. G is a pair (V, E) of sets such that $E \subseteq V^2$ and E does not intersect the diagonal. This will allow us to write an edge e as (u, v) where e goes from u to v.

Definition 8.1. A **time-extended graph** is a directed graph $G = (V, E)$ and a set $\mathcal{N}$ with functions $q : V \to \mathcal{N}$ and $t : V \to \mathbb{Z}$ such that for each $n \in \mathcal{N}$, t is injective on $q^{-1}(n)$ and such that for every u and v with $q(u) = q(v)$, $t(v) = t(u) + 1$ if and only if there is an edge $(u, v) \in E$. We call $\mathcal{N}$ the **set of names**, $q(v)$ the **name** of v, and $t(v)$ the **time step** of v.

An example of a time-extended graph is below.



(1)

$$q : \begin{array}{lcl} a_0, a_1, a_2, a_3 & \longmapsto & a \\ b_0, b_1, b_2 & \longmapsto & b \\ c_0, c_1, c_2, c_3 & \longmapsto & c \\ d_0, d_1 & \longmapsto & d \end{array}$$

$$t : \begin{array}{lcl} a_0, b_0, c_0, d_0 & \longmapsto & 0 \\ a_1, b_1, c_1, d_1 & \longmapsto & 1 \\ a_2, b_2, c_2 & \longmapsto & 2 \\ a_3, c_3 & \longmapsto & 3 \end{array}$$

The name function q keeps track of the same object at different time steps. For example, an edge from a_0 to a_1 represents the fact that a can store information over time. In other words, a can receive information from itself one time step in the past and can send information to itself one time step the future. Time-extended graphs let us model a system that changes over time. Not only can edges drop in and out, but named objects can drop in and out as well. The biggest advantage of using time-extended graphs is that they allow us to represent relativistic effects. The system we want to model will not always have an objective measure of time by which all clocks can be synchronized, but with our definition of time extended graphs, we do not have trouble modeling this situation. To construct the graph, we just need to know e.g. that if a signal leaves object c at time 2 on its clock, then it will arrive at object d at time 0 on d's clock.

This suggests that applicability might exist beyond network segments that feature sufficiently low round trip times to achieve real-time feedback.

Of course, even for a small number of named objects, constructing such a time-extended graph by hand would be a formidable task. For the application of space communications that the authors are interested in, it is expected that orbit analysis software will be used to construct the graph. Some simple code would be able to convert data exported from a program such as SOAP to a time-extended graph.

Multicast Indexing Sheaves

For each vertex v in a graph G, let $\mathcal{A}(v)$ be the set of vertices adjacent to v, this is sometimes also called the **link** of the vertex v. More precisely,

$$\mathcal{A}(v) = \{u \mid (u, v) \in E \text{ or } (v, u) \in E\}.$$

Definition 8.2. The **universal multicast (indexing) sheaf** $\mathcal{M}$ on G is defined on vertices v by

$$\mathcal{M}(v) = \mathscr{P}(\mathcal{A}(v)),$$

the powerset of $\mathcal{A}(v)$, and on edges e by

$$\mathcal{M}(e) = \{0, 1\}.$$

On restrictions $u \leq e$, where $e = (u, v)$ or $e = (v, u)$,

$$\mathcal{M}(u \leq e) : \quad S \longmapsto \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{otherwise.} \end{cases}$$

Each element of $\mathcal{M}(v)$ can be interpreted as a list of the vertices that v is simultaneously in communication in a given scenario. The elements of $\mathcal{M}(e)$ represent an edge being "on", 1, or "off", 0. A section of $\mathcal{M}$ amounts to a coherent assignment of 'on' and 'off' to each edge, and for each vertex, a list of which vertices it is in communication with.

Remark 8.3. We call $\mathcal{M}$ an indexing sheaf because it can be used to index more complicated forms of networking [24].

The multicast sheaf is universal because it organizes other indexing subsheaves. We remind the reader of this concept now.

Definition 8.4. A **subsheaf** of a sheaf $\mathcal{F} : P_G \to \mathbf{Set}$ is a functor $\mathcal{H} : P_G \to \mathbf{Set}$ such that $\mathcal{H}(g) \subseteq \mathcal{F}(g)$ on each $g \in V \cup E$ and $\mathcal{H}(u \leq e)$ is the restriction of $\mathcal{F}(u \leq e)$ to $\mathcal{H}(u)$ on each incidence relation $u \leq e$.

The universality of the multicast sheaf can be summarized as follows: every subsheaf that agrees with $\mathcal{M}$ on edges can be used as such an indexing sheaf. A particular subsheaf we are interested in is unicast communication.

Definition 8.5. The **unicast (indexing) sheaf** $\mathcal{U} : P_G \to \mathbf{Set}$. This is a subsheaf of $\mathcal{M}$ defined on edges by

$$\mathcal{U}(e) = \mathcal{M}(v) = \{0, 1\}$$

and on vertices by

$$\mathcal{U}(v) = \left\{ S \in \mathcal{M}(v) \;\middle|\; |\{x \in S \mid q(x) \neq q(v)\}| \leq 1 \right\}.$$

In words, $\mathcal{U}(v)$ is the set of sets S in $\mathscr{P}(\mathcal{A}(v))$ such that the subset of S on the elements that have a different name than v has cardinality less than or equal to 1.

Identifying the Sections of a Sheaf of Sets

The Combinatorial Approach—The most straightforward approach to finding the sections of a cellular sheaf $\mathcal{F} : P_G \to \mathbf{Set}$ is a combinatorial one. We simply apply the maps δ^+ and δ^- to each element of $\prod_{v \in V} \mathcal{F}(v)$ and make note of the elements for which the two maps agree. In practice, however, the set $\prod_{v \in V} \mathcal{F}(v)$ can be quite large. Even under the mild assumption that $|\mathcal{F}(v)| \geq 2$ for each v we get an exponential search space for sections because

$$\left| \prod_{v \in V} \mathcal{F}(v) \right| \geq 2^{|V|}.$$

This exponential growth means that the straightforward method of checking each element quickly becomes infeasible as G gets large. As such, we will need a different approach for even moderately large graphs.

Sections and Equalizers

We will use the notation $\mathbb{F} : \mathbf{Set} \to \mathbb{R}\text{-}\mathbf{Vect}$ for the functor that sends each set X to the free $\mathbb{R}$-vector space $\mathbb{R}^{|X|}$. The functor $\mathbb{F}$ sends each map of sets $f : X \to Y$ to the linear map $\mathbb{F}(f) : \mathbb{R}^{|X|} \to \mathbb{R}^{|Y|}$ given by

$$\sum_{x \in X} c_x \vec{e}_x \longmapsto \sum_{x \in X} c_x \vec{e}_{f(x)}.$$

The following result illustrates a method for performing a linear relaxation of this problem. Its proof is immediate because sections of the original sheaf are sections of its linearization.

Proposition 8.6. *Let $\mathcal{F} : P_G \to \mathbf{Set}$ be a cellular sheaf on a directed graph. Let $\mathcal{F}(G)$ be the equalizer of the diagram*

$$\begin{tikzcd} \prod_{v \in V} \mathcal{F}(v) \arrow[r, shift left, "\delta^+"] \arrow[r, shift right, "\delta^-"'] & \prod_{e \in E} \mathcal{F}(e) \end{tikzcd} \qquad (2)$$

and let K be the equalizer of the diagram

$$\begin{tikzcd} \mathbb{F}\Big(\prod_{v \in V} \mathcal{F}(v) \Big) \arrow[r, shift left, "\mathbb{F}(\delta^+)"] \arrow[r, shift right, "\mathbb{F}(\delta^-)"'] & \mathbb{F}\Big(\prod_{e \in E} \mathcal{F}(e) \Big) \end{tikzcd}$$

we get by applying the free functor to diagram (2). Then $\mathbb{F}(\mathcal{F}(G)) \subseteq K$. We note that K can be interpreted as the kernel of the linear map $\mathbb{F}(\delta^+) - \mathbb{F}(\delta^-)$

Certainly every linear combination of sections of $\mathcal{F}$ is an element of K, but not every element of K can be regarded as a section. Figure 2 illustrates how this linear relaxation can produce extraneous sections.

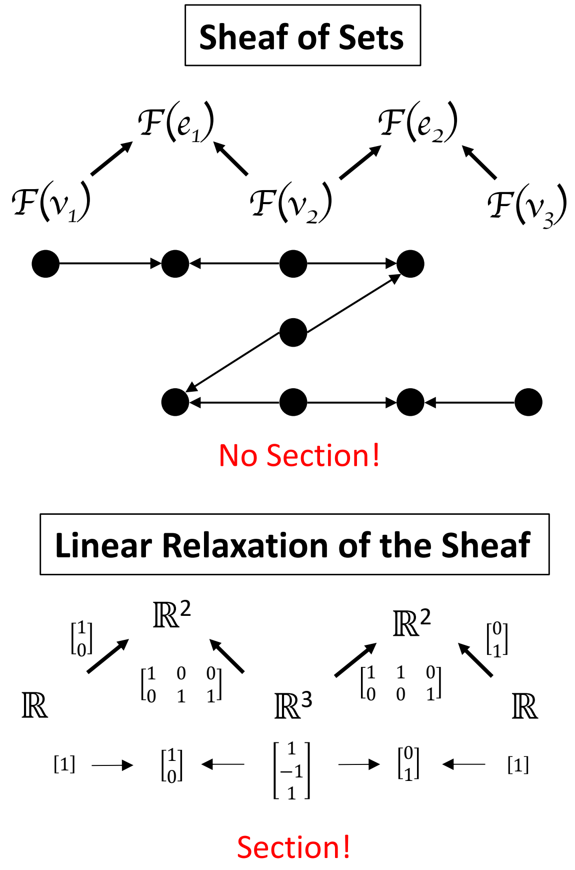


Figure 2. Linear Relaxation Introduces Extra Sections

One way to circumvent the problem presented in Figure 2 is to consider only those vectors in the kernel K that have 0's and 1's as entries. One can interpret this as identifying the integer solutions inside our linear relaxation.

Proposition 8.7. *A vector $\vec{x}$ in $\mathbb{F}\big(\prod_{v \in V} \mathcal{F}(v) \big)$ is a section of $\mathcal{F}$ if and only if $\vec{x}$ is in the kernel K and every entry of $\vec{x}$ is in $\{0, 1\}$.*

Using Gradient Descent to Find a Basis of Sections—Finding the kernel of a linear map is a much more computationally feasible problem than checking a condition on each element of a set. The problem is that the basis we get when we compute the kernel does not have to be a basis of sections. We should expect it to be a basis of superpositions of sections.

Gradient descent can be used to find a basis of actual sections from a basis of superpositions. Proposition 8.7 tells us enough about what an actual section should look like that we can write down a good cost function which penalizes superpositions of sections that are far from actual sections. We will now turn to the description of the cost function for our gradient descent.

Let $m = \Big|\prod_{v\in V}\mathcal{F}(v)\Big|$ be the dimension of the ambient space $\mathbb{F}\Big(\prod_{v\in V}\mathcal{F}(v)\Big)$, and let n be the dimension of the kernel K. The cost function c will be a composition of functions

$$\begin{tikzcd}
\mathbb{R}^n \arrow[r, "f"] \arrow[rrr, bend left, "c"] & \mathbb{R}^m \arrow[r, "g"] & \mathbb{R}^m \arrow[r, "h"] & \mathbb{R}
\end{tikzcd}$$

The map f is a parameterization of the subspace K of $\mathbb{F}\Big(\prod_{v\in V}\mathcal{F}(v)\Big)$. Let $\{\vec{u}_j\}_{j=1}^n$ be a basis for K. Then every vector in K can be expressed as a sum $\sum_{j=1}^n a_j\vec{u}_j$. So we define $f : \mathbb{R}^n \to \mathbb{R}^m$ as

$$f : (x_j)_{j=1}^n \quad\longmapsto\quad \left(\sum_{j=1}^n x_j u_{ji}\right)_{i=1}^m,$$

where u_{ji} is the ith coordinate of $\vec{u}_j$.

The map $g : \mathbb{R}^m \to \mathbb{R}^m$ is the defined as

$$g : (u_i)_{i=1}^m \longmapsto \left(u_i^2(u_i-1)^2\right)_{i=1}^m.$$

So at coordinate i, $g\left((u_i)_{i=1}^m\right)$ is a positive measure of how far u_i is from either 0 or 1.

Finally, the map $h : \mathbb{R}^m \to \mathbb{R}$ is defined as the sum

$$h : (x_i)_i \mapsto \Sigma_i x_i.$$

Since the coordinates of objects in the image of g are always positive, h is a good measure of distance from a section.

To do gradient descent, we need to calculate the partial derivatives of the cost function $c = h \circ g \circ f$. By the multivariate chain rule,

$$\frac{\partial c}{\partial x_j}(\vec{x}) = \sum_{k=1}^m \frac{\partial h}{\partial z_k}\Big(g\big(f(\vec{x})\big)\Big) \sum_{i=1}^m \frac{\partial g_k}{\partial y_i}\big(f(\vec{x})\big)\frac{\partial f_i}{\partial x_j}(\vec{x}).$$

Note that $\frac{\partial g_k}{\partial y_i}$ is only nonzero when $k = i$. Therefore

$$\frac{\partial c}{\partial x_j}(\vec{x}) = \sum_{k=1}^m \frac{\partial h}{\partial z_k}\Big(g\big(f(\vec{x})\big)\Big)\frac{\partial g_k}{\partial y_k}\big(f(\vec{x})\big)\frac{\partial f_k}{\partial x_j}(\vec{x}).$$

We further note that $\frac{\partial h}{\partial z_k}$ is a constant function 1 for every k. Therefore

$$\frac{\partial c}{\partial x_j}(\vec{x}) = \sum_{k=1}^m \frac{\partial g_k}{\partial y_k}\big(f(\vec{x})\big)\frac{\partial f_k}{\partial x_j}(\vec{x}).$$

Using the above computation of the gradient, we implemented a simple adaptive gradient descent algorithm. The algorithm sets m to an initial value of 0.1 and sets the input vector $\vec{x}$ to $\vec{x} - m\nabla c(\vec{x})$ at each step where this change results in a decrease in the cost function at $\vec{x}$. If the change results in an increase in the value of the cost function, we do not implement the change, and we reset m to be half of its former value. If the percentage decrease in the cost function is below some threshold, we reset m to be 1.5 times its former value. The pseudocode for this algorithm is included below.

```
def descent(input, cost_function,
    ↪ cost_gradient, threshold,
    ↪ max_steps, rel_threshold):
    multiplier = 0.1
    step_input_1 = input
    cost_1 = cost_function(step_input_1)
    gradient = cost_gradient(step_input_1
        ↪ )
    for step in range(max_steps):
        if cost_1 <= threshold:
            return step_input_1
        step_input_2 = step_input_1 -
            ↪ multiplier*gradient
        cost_2 = cost_function(
            ↪ step_input_2)
        if cost_1 < cost_2:
            multiplier = multiplier/2
        elif (cost_1-cost_2)/cost_1 <
            ↪ rel_threshold:
            multiplier = multiplier*1.5
            step_input_1 = step_input_2
            cost_1 = cost_2
            gradient = cost_gradient(
                ↪ step_input_2)
        else:
            step_input_1 = step_input_2
            cost_1 = cost_2
            gradient = cost_gradient(
                ↪ step_input_2)
    return step_input_1
```

Since we know the form of the absolute minima, we can use a relatively large value for the threshold variable and then round each entry to the nearest integer. This will reduce the number of computationally expensive gradient descent steps that we have to do. While the rounding approach will occasionally get an incorrect result, we can quickly check whether the result is correct by applying the map $\mathbb{F}(\delta^+) - \mathbb{F}(\delta^-)$ to it. If it is incorrect, we just discard the vector and try again.

We find a basis B by choosing a random vector in $\mathbb{R}^n$ and applying the gradient descent algorithm to it. Starting with $B = \emptyset$, any time the algorithm converges to a section in the kernel, we check to see if adding the section to B results in a linearly independent set. If so, then we add the vector to B. If not, we discard the vector. If $|B| < n$, we choose a new random vector in $\mathbb{R}^n$ and start the process over again.

Whereas the combinatorial approach to this problem has big

O complexity $\mathcal{O}(2^m)$, the algorithm we propose here has big O complexity $\mathcal{O}(mn)$. The value max_steps may be large, which could result in a complexity growth rate proportional to 2max_stepsmn. This coefficient tends to not be a very tight upper bound however, since in the examples we ran, gradient descent often achieves the threshold long before reaching step max_steps.

The gradient descent followed by rounding approach could potentially be improved by selecting an interval of uncertainty such as $[.3, .7]$, stopping the gradient descent even sooner, rounding vector coordinates that are outside of the interval, and trying both 0 and 1 for the vector coordinates that are in the interval. When we have multiple vector coordinates that lie within the interval of uncertainty, we would need to try all possible combinations of 0's and 1's for these coordinates. With this approach, we may only need to run gradient descent a small number of times to get to a point where fewer than 10 coordinates are within the interval of uncertainty. Then we would only have 2^{10} combinations to check, which would likely be computationally faster than continuing gradient descent. We have not implemented this approach yet, but we believe that this combination of combinatorial and gradient descent methods will result in a fast algorithm. We would simply be using gradient descent to reduce the size of the combinatorial problem we are working with to much more manageable pieces.

Example Unicast Sheaf Sections—We implemented the above method to find sections of the unicast indexing sheaf $\mathcal{U}$ on graph (1). This graph has 13 vertices and 17 edges.

We found that the gradient descent method for finding sections was very effective. We set threshold to 0.15, max_steps to 500, and rel_threshold to 0.01. For a trial of 1,000 random vectors, the method of using gradient descent followed by rounding converges to an actual section 98.5 percent of the time. These sections correspond to absolute minima of the cost function. Indeed, the cost function for a section is 0. The other 1.5 percent of vectors either got caught in a local minimum that was not absolute, converged to the absolute minimum too slowly, or reached the threshold, but still rounded to the wrong value.

With the above parameters and the basis search method described in the previous section, we were able to find a basis of $n = 18$ vectors with an average of 2.5 seconds. There is plenty of room for improvement of this time as well. The combination of the gradient descent and the combinatorial guess-and-check approaches described in the previous section would likely speed things up even more. Also, since this code was written in Python, writing in a more elementary language would result in significant time improvements. Of course, using a better computer would also speed things up.

As a point of comparison, when we ran the purely combinatorial approach, after 25 minutes, our computer had not found a single section, even though there were many. Consequently, the benefits of the gradient descent approach are already evident even in this relatively small example.

Implementing Gradient Descent for Networking Coding

The implementation of gradient descent to make network coding decisions is still a work in progress. We outline the general idea of our approach here, but much of this has not been completed yet.

Our goal is to use the basis of sections for a unicast or multicast sheaf to perform gradient descent on the possible ways to send linear combinations of data through our network. We will be applying a linear map at each vertex, which will convert the data entering through input edges to data leaving the output edges. More explicitly, the variables of our gradient descent should consist of

1. Coefficients of section superpositions, and
2. Entries of the matrices corresponding to the linear maps above each vertex.

The cost function for our gradient descent will consist of two parts. First, we will produce a large list of different artificially generated signals. We will initialize the variables, feed each element of the list into our source vertices, collect the signals that come out of the sink vertices, and compute the squared difference between the inputs and outputs. Summing these differences up over all the elements of our list will give us a cost function that will force the outputs to agree with the inputs to the extent possible. If we perform this for a large enough list of data and our able to reach a cost of 0, then we can assume that the network is actually preserving our data. Even though even pieces of the messages will likely not be recoverable at intermediate edges, at the outputs, it will combine to reproduce the input signals.

The second part of our cost function will need to penalize superpositions of sections that are far from actual sections. This part of the cost function will likely take the same form as the cost function described in Section 8. The hope is that by forcing the result to converge to a section, we will end up with a meaningful description of a route through our network that obeys the networking rules specified by our indexing sheaf.

Likely we will want to initially place little weight on the second part of the cost function, but increase the weight placed on this part as the number of gradient descent steps increase.

9. Conclusion and Future Work

In this paper, we extended our previous applications of sheaves to network theory. The improved modeling techniques – for the networks themselves as well as routing over them – can lead to improved algorithms as well as improved systems of algorithms by modeling how they interact.

As more fundamental networking structures – such as trees – get represented as sheaves and cosheaves, new and different tools can be brought to bear to study networks. As an example, we were able to leverage our edges cosheaf to bring zigzag persistent homology to networks. In the future, more tools from traditional networking can be recast to operate in the context of sheaves and then expanded to DTN generally. This includes approaches to routing, and in particular the means to bring different routing algorithms to bear on one network.

Once sheaf theoretic structures are in place, optimization becomes possible, as the space of possibilities is more clearly defined. Here we were able to leverage gradient descent to explore the space of sections of our multicast sheaves and to explore network coding. In the future, we hope to see other machine learning algorithms – such as neural networks or other cognitive systems – drawing information from sheaf theoretic data structures for routing and network optimization purposes.

Future Work

We conclude with some thoughts on possible future work in the various areas we have explored.

1. The tree sheaves of Section 4 and the pullbacks of sheaves of Section 5 should together be able to describe a combined use of Dijkstra's algorithm and the Bellman–Ford algorithm in finding a single path. Implementing such an example in a sheaf-theoretic setting may lead to new insight into how algorithms used in various parts of a network can be "glued."
2. As shown in Figure 1, the ability to model networks of large scales exists. It remains to implement many diverse scenarios to test them against the same sheaf-based framework.
3. A follow-on is to use these tools to determine natural boundaries and substructures of delay tolerant networks. By identifying analogous network segments and types, we can design efficient structures for autonomous systems.
4. As we explore structures for DTN, summary statistics for understanding how dynamic networks are changing will become relevant to simplify inputs to machine learning algorithms. While the natural thing might be to construct functions on the edges representing different statistics, extending to sheaves – in a way similar to the weight sheaf example from Section 2 – may be useful for determining what information can be localized at different nodes. In addition, the fact that summary statistics can be represented as vectors of data leads to some natural implementation possibilities within PySheaf [10]. Studying these sheaves may provide useful structures and demos in the future.
5. The initial approach to zigzag persistence was to start with a weighted digraph and take interlevelset persistence, but we found that it failed to be stable with respect to the network distance of [26], [32]. In the case of a dynamic digraph, there doesn't currently exist an appropriate "temporal" network distance, so we cannot even conjecture a stability theorem in this setting. This leads to the open question of what the right notion of temporal network distance should be. On the computational side, some work has been done on efficiently calculating persistent path homology, but we need to develop algorithms for zigzag path homology in order to compute their barcodes on simulations.
6. A full, proven implementation of the network coding sheaf approach remains open, and as illustrated in Figure 2, there is work to be done to make sure that the approach yields sections. This would likely include a reformulation and tweaks to the mappings, but it would come with an immediate payoff, as it is readily implementable and hence suitable for testing in network simulators.

References

[1] NASA, "New solar system internet technology debuts on the international space station," Jun. 2016. [Online]. Available: https://www.nasa.gov/feature/new-solar-system-internet-technology-debuts-on-the-international-space-station

[2] N. Mohanta, "How many satellites are orbiting the earth in 2021?" May 2021. [Online]. Available: https://www.geospatialworld.net/blogs/how-many-satellites-are-orbiting-the-earth-in-2021/

[3] NASA, "Delay/disruption tolerant networking," Sept. 2020. [Online]. Available: https://www.nasa.gov/directorates/heo/scan/engineering/technology/disruption_tolerant_networking

[4] J. A. Fraire, P. Madoery, S. Burleigh, M. Feldmann, J. Finochietto, A. Charif, N. Zergainoh, and R. Velazco, "Assessing Contact Graph Routing Performance and Reliability in Distributed Satellite Constellations," Jul. 2017. [Online]. Available: https://www.hindawi.com/journals/jcnc/2017/2830542/

[5] M. Demmer and K. Fall, "Dtlsr: Delay tolerant routing for developing regions," in *Proceedings of the 2007 workshop on Networked systems for developing regions*, ser. NSDR '07. New York, NY, USA: Association for Computing Machinery, 2007. [Online]. Available: https://doi.org/10.1145/1326571.1326579

[6] S. Grasic, E. Davies, A. Lindgren, and A. Doria, "The evolution of a dtn routing protocol - prophetv2," in *Proceedings of the 6th ACM Workshop on Challenged Networks*, ser. CHANTS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 27–30. [Online]. Available: https://doi.org/10.1145/2030652.2030661

[7] A. Hylton, R. Short, R. Green, and M. Toksoz-Exley, "A mathematical analysis of an example delay tolerant network using the theory of sheaves," in *2020 IEEE Aerospace Conference*, 2020, pp. 1–11.

[8] R. Short, R. Green, M. Moy, and B. Story, "What type of graph is a contact graph?" NASA, Glenn Research Center, Cleveland OH 44135, USA, Technical Memorandum Upcoming, 2021.

[9] R. Short, A. Hylton, R. Cardona, R. Green, G. Bainbridge, M. Moy, and J. Cleveland, "Towards sheaf theoretic analyses for delay tolerant networking," in *2021 IEEE Aerospace Conference (50100)*, 2021, pp. 1–9.

[10] M. Robinson, C. Capraro, and B. Praggastis, "The pysheaf library," 2016. [Online]. Available: https://github.com/kb1dds/pysheaf

[11] R. Green, R. Cardona, J. Cleveland, J. Ozbolt, A. Hylton, R. Short, and M. Robinson, "Dude where's my stars: A novel topologically justified approach to star tracking," in *2021 IEEE Aerospace Conference*, 2021.

[12] M. Robinson, "Sheaves are the canonical data structure for sensor integration," *Information Fusion*, vol. 36, pp. 208–224, 2017.

[13] R. G. Swan, *The Theory of Sheaves*. University of Chicago Press, 1964.

[14] G. E. Bredon, *Sheaf theory*, 2nd ed., ser. Graduate Texts in Mathematics. Springer-Verlag, 1997, vol. 170.

[15] M. Kashiwara and P. Schapira, *Sheaves on Manifolds: With a Short History.Les débuts de la théorie des faisceaux. By Christian Houzel*. Springer Science & Business Media, 2013, vol. 292.

[16] M. Robinson, "Hunting for foxes with sheaves," *Notices of the American Mathematical Society*, vol. 66, pp. 661–676, May 2019.

[17] K. Baclawski, "Whitney numbers of geometric lattices," in *Surveys in Applied Mathematics*. Elsevier, 1976, pp. 103–116.

[18] J. Curry, "Functors on posets left kan extend to cosheaves: an erratum," *arXiv preprint arXiv:1907.09416*, 2019.

[19] ——, "Sheaves, cosheaves and applications," *arXiv*, 2013. [Online]. Available: http://arxiv.org/abs/1303.3255

[20] S. Awodey, *Category Theory*. New York: Oxford University Press Inc., 2012.

[21] S. Mac Lane, *Categories for the Working Mathematician*. New York: Springer-Verlag New York, Inc., 1998.

[22] A. D. Shepard, "A cellular description of the derived category of a stratified space," Ph.D. dissertation, Brown University, 1985.

[23] M. Moy, R. Cardona, R. Green, J. Cleveland, A. Hylton, and R. Short, "Path optimization sheaves," 2020. [Online]. Available: https://arxiv.org/abs/2012.05974

[24] G. Bainbridge, A. Hylton, and R. Short, "Directional cellular sheaves for multicast network routing," N.D., unpublished.

[25] G. Carlsson and V. de Silva, "Zigzag persistence," *Foundations of Computational Mathematics*, vol. 10, no. 4, pp. 367–405, Aug. 2010. [Online]. Available: https://doi.org/10.1007/s10208-010-9066-0

[26] S. Chowdhury and F. Mémoli, "Persistent path homology of directed networks," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018, p. 1152–1169.

[27] A. Grigor'yan, Y. Lin, Y. Muranov, and S.-T. Yau, "Homotopy theory for digraphs," *arXiv e-prints*, p. arXiv:1407.0234, Jul. 2014.

[28] H. Bjerkevik, "On the stability of interval decomposable persistence modules," *Discrete & Computational Geometry*, vol. 66, Jul. 2021.

[29] M. Bakke Botnan and M. Lesnick, "Algebraic Stability of Zigzag Persistence Modules," *arXiv e-prints*, p. arXiv:1604.00655, Apr. 2016.

[30] V. De Silva, E. Munch, and A. Patel, "Categorified reeb graphs," *Discrete Comput. Geom.*, vol. 55, no. 4, p. 854–906, Jun. 2016. [Online]. Available: https://doi.org/10.1007/s00454-016-9763-9

[31] T. Ho and D. Lun, *Network Coding: An Introduction*. Cambridge University Press, 2008.

[32] G. Carlsson, F. Mémoli, A. Ribeiro, and S. Segarra, "Hierarchical quasi-clustering methods for asymmetric networks," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2. Bejing, China: PMLR, Jun. 2014, pp. 352–360. [Online]. Available: https://proceedings.mlr.press/v32/carlsson14.html

BIOGRAPHIES

Robert Short earned his PhD in mathematics from Lehigh University in 2018. He worked as a Visiting Assistant Professor of Mathematics at John Carroll University until he joined the Secure Networks, System Integration and Test Branch at NASA Glenn Research Center in 2020. His research interests lie in the intersection of abstract mathematics and real world applications. Currently, his focus is on the foundations of networking theory and how to efficiently route data through a network using local information.

Alan Hylton should probably be designing tube audio circuits, but instead directs Delay Tolerant Networking (DTN) research and development at the NASA Glenn Research Center, where he is humbled to work with his powerful and multidisciplinary team. His formal education is in mathematics from Cleveland State University and Lehigh University, and he considers it his mission to advocate for students. Where possible, he creates venues for mathematicians to work on applied problems, who add an essential diversity to the group.

Jacob Cleveland is a Senior studying Mathematics and Computer Engineering at the University of Nebraska at Omaha. They joined the Secure Networks, System Integration and Test Branch as a Pathways Intern at NASA Glenn Research Center in 2020. Since joining, they have performed several research projects applying pure mathematics to engineering problems such as networking in space, star tracking, and artificial neural networks.

Michael Moy is pursuing a PhD in mathematics at Colorado State University, having completed his master's there in 2021. His research is focused on applied topology. During the summers of 2020 and 2021, he worked as an intern at NASA through the SCaN Internship Project. His research areas at NASA have included machine learning and mathematical approaches to networking.

Robert Cardona is a PhD student in applied topology. He studied computer engineering and mathematics before going on to work as a software developer. He then obtained a masters at Freie Universität Berlin and continued on to study applied topology at Albany. He worked as an intern at NASA during the summer of 2021.

Robert Green is a 2nd year mathematics PhD student at the University at Albany. He is studying topics including applications of Topological Data Analysis (TDA) and Graph Theory to space networking problems under Justin Curry. He previously completed his undergraduate and masters degrees in mathematics from American University where he worked with Michael Robinson on topics such as TDA and Signal Processing.

Justin Curry is an Assistant Professor of Mathematics and Statistics at the University at Albany, SUNY. Before arriving at Albany in 2017, he was a Visiting Assistant Professor at Duke University. Professor Curry earned his PhD in mathematics from the University of Pennsylvania in 2014, under the direction of Robert Ghrist. His research interests include the use of category theory in applied mathematics, with particular emphasis on applied sheaf theory, and inverse problems in topological data analysis (TDA).

Brendan Mallery is a PhD student studying mathematics at Tufts University. Previously he received a Masters in Mathematics from the University at Albany, SUNY in 2020, and a Bachelors in Mathematics and Chemistry from Bowdoin College in 2018. His research interests include geometric group theory, optimal transport and applied sheaf theory.

Gabriel Bainbridge earned his PhD in mathematics from the Ohio State University in 2021. During the summers of 2020 and 2021, he worked as an intern at NASA through the SCaN Internship Project. His research interests include applied category theory, networking theory, and machine learning techniques.

Zander Memon Zander Memon is a senior mathematics and economics major at American University. He worked as an applications of pure mathematics intern at NASA Glenn Research Center in 2021, and has been a research assistant in both the mathematics and economics departments at American University since 2019. Zander's research interests include applied mathematics, algebraic topology, game theory, and topological data analysis. After graduation he plans to pursue graduate school in mathematics.